

NB Toolbox

Reference Manual

Kenneth Sæterhagen Paulsen

January 6, 2023

Abstract

The NB toolbox is an object oriented toolbox for managing data, creating graphics and doing econometrics.¹ ² This reference manual will provide a general description of the main classes and functions of this toolbox. Some examples will be given along the way. The last section will give an extensive description of all classes, methods and functions found in this toolbox.

¹Developers; Kenneth Sæterhagen Paulsen, Per Bjarne Bye, Eyo Alexander Ildahl Herstad, Henrik Halvorsen Hortemo, Tobias Ingebrigtsen, Atle Loneland and Andreas Haga Raavand.

²Special thanks to; Knut Are Aastveit, SeHyoun Ahn, André Kallåk Anundsen, Andrew Binning, Christian Bjørland, Tuva Marie Fastbø, Karsten R. Gerdrup, Erling Motzfeldt Kravik, Junior Maih, Sara Skjeggestad Meyer, Yasin Mimir, Ørjan Robstad and Pål Bergset Ulvedal for comments and suggestions.

Contents

1 Setup toolbox	7
1.1 System requirements	7
1.2 Resolution and screen setup	7
2 Theoretical foundation	7
3 Get help on classes and functions	7
4 Dates	8
4.1 Dates as strings	8
4.2 Dates as objects	8
4.3 Examples	9
5 Data management	9
5.1 Time series data	9
5.1.1 Read data from excel	9
5.1.2 Read data from a .mat file	10
5.1.3 Initialize an data object with MATLAB types	11
5.1.4 Transformation of the data	12
5.1.5 Creating new variables	12
5.1.6 Converting data to a new frequency	13
5.1.7 Merging data	15
5.1.8 Writing data to Excel	16
5.1.9 Writing data to other file formats	17
5.1.10 Converting to other data types (objects)	18
5.1.11 Mathematical operators	18
5.1.12 Logical operators	18
5.1.13 Other useful methods	19
5.1.14 The nb_math_ts class	19
5.1.15 Examples	20
5.2 Cross sectional data	20
5.2.1 Read data from excel	20
5.2.2 Read data from a .mat file	21
5.2.3 Initialize an data object with MATLAB types	21
5.2.4 Transformation of the data	22
5.2.5 Creating new variables and types	22
5.2.6 Merging data	23
5.2.7 Writing data to Excel	23
5.2.8 Writing data to other file formats	24
5.2.9 Converting to other data types (objects)	24
5.2.10 Mathematical operators	24
5.2.11 Logical operators	25
5.2.12 Other useful methods	25
5.2.13 Examples	25
5.3 Dimensionless data	25
5.3.1 Read data from excel	25
5.3.2 Read data from a .mat file	27
5.3.3 Initialize an data object with MATLAB types	27
5.3.4 Transformation of the data	27
5.3.5 Creating new variables	28
5.3.6 Merging data	28
5.3.7 Writing data to Excel	29
5.3.8 Writing data to other file formats	30

5.3.9	Converting to other data types (objects)	30
5.3.10	Mathematical operators	30
5.3.11	Logical operators	31
5.3.12	Other useful methods	31
5.3.13	Examples	31
5.4	Import multi-paged data	32
5.4.1	Excel spreadsheet	32
5.4.2	A note on real-time data	32
5.5	Create a link to the data source	32
6	Graphics	32
6.1	Time-series data	32
6.1.1	The method graph()	33
6.1.2	The method graphSubPlots()	35
6.1.3	The method graphInfoStruct()	35
6.1.4	Set the spacing between the x-axis tick marks	38
6.1.5	Set the frequency of the x-axis tick marks	38
6.1.6	Set the start date of the x-axis tick marks	38
6.1.7	Set how to interpret the data	39
6.1.8	Set the alignment of the x-axis tick mark labels	39
6.1.9	Set the location of the x-axis tick marks	40
6.1.10	How to interpret missing data	40
6.1.11	Creating a fan chart	41
6.1.12	Adding highlighted areas	42
6.2	Cross-sectional data	43
6.2.1	The method graph()	44
6.2.2	The method graphSubPlots()	46
6.2.3	The method graphInfoStruct()	46
6.2.4	Multi line x-axis tick mark labels	48
6.3	Dimensionless data	50
6.3.1	The method graph()	50
6.3.2	The method graphSubPlots()	52
6.3.3	The method graphInfoStruct()	52
6.4	Shared functionality	52
6.4.1	Plot against different axes	52
6.4.2	Set properties of the y-axes	53
6.4.3	Colors	53
6.4.4	Edit the legend	55
6.4.5	Adding a fake legend	55
6.4.6	Adding labels	56
6.4.7	Set the graph style	57
6.4.8	Adding horizontal lines	58
6.4.9	Adding vertical lines	58
6.4.10	Normalize the font units	59
6.4.11	Save the graph(s) to a file	59
6.4.12	Writing the data of the graph(s) to Excel	60
6.5	nb_graph_ts examples	60
6.6	nb_graph_cs examples	60
6.7	nb_graph_data examples	60
6.8	Adding annotations	61
6.8.1	More examples	61
6.9	Creating subplots	61
6.10	Create a package of graphs	62
6.10.1	Adding figure title and footer	62

6.10.2	Initializing a graph package	64
6.10.3	Adding an graph object to the graph package	65
6.10.4	Print the graph(s) to a pdf file	65
6.10.5	Writing data of the graph package to Excel	65
6.10.6	Full example script	66
6.11	Basic plot classes and functions	66
6.11.1	Examples	68
7	Basic Econometrics	68
7.1	Autocorrelation	68
7.2	Unit root tests	68
7.3	Cointegration tests	68
7.4	Information Criterion	69
7.5	Frequency zero spectrum estimation	69
7.6	Kernel functions	69
7.7	Bootstrapping	69
8	Single equation, time-series	69
8.1	OLS	70
8.2	TSLS	70
8.3	Estimation results	70
8.4	Automatic model selection	70
8.5	Test statistics	71
8.5.1	ARCH (Autoregressive conditional heteroskedasticity)	71
8.5.2	Autocorrelation	71
8.5.3	Breusch-Pagan test for heteroskedasticity	71
8.5.4	Chow test for a structural break	71
8.5.5	Durbin-Wu-Hausman test for exogeneity	71
8.5.6	F-test	71
9	Expression model	72
9.1	OLS	72
10	Step ahead model	72
11	ARIMA	73
11.1	Maximum likelihood	74
11.2	Hannan-Rissan	74
12	Error-correction model (ECM)	75
13	VAR	75
13.1	OLS	76
13.2	Ridge	76
13.3	LASSO	76
13.4	B-VAR	76
13.5	Empirical baysian	76
13.6	Time varying parameter bayesian estimation	77
13.7	Identification	77
13.8	Test the VAR	78
13.8.1	Test for stationarity	78
13.8.2	LM-test for autocorrelation	78
13.8.3	Ljung-Box test for autocorrelation	78
13.8.4	Granger causality	78

14 VAR with missing observations	79
14.1 MO-VAR	79
14.1.1 Maximum likelihood	79
14.1.2 Bayesian	80
14.2 MF-VAR	80
14.2.1 Maximum likelihood	80
14.2.2 Bayesian	81
14.3 Time varying parameter bayesian estimation	81
14.4 Variables observed at different frequencies	81
14.4.1 Change in frequency	81
14.4.2 Observed at different frequencies at the same time	81
14.5 Adding measurement restrictions	81
15 MIDAS	82
16 Factor models using principal component	82
16.1 Principal component	82
16.2 Single equation factor model	83
16.3 Step ahead factor model	83
16.4 FA-VAR	84
17 Dynamic factor model	84
17.1 Expected maximum likelihood	85
17.2 Time varying parameter bayesian estimation	85
18 Missing observations	85
19 DSGE	86
19.1 Standard 1st order solution	86
19.2 Optimal monetary policy	88
19.3 Model file	89
19.4 Steady-state file	89
19.5 Calibration	89
19.6 Estimation	89
19.6.1 Mode estimation	89
19.6.2 Sampling	90
19.6.3 System priors	91
19.7 Filtering	92
19.8 DSGE model with break-point	92
19.8.1 Forecast	92
19.8.2 IRF	93
19.8.3 Filtering	93
19.9 Optimal simple rules	93
19.9.1 Commitment	93
19.9.2 Discretion	94
19.9.3 Calculate loss	95
20 Analysis	95
20.1 Impulse response functions	95
20.2 Shock decomposition	95
20.3 Forecast error variance decomposition (FEVD)	95
20.4 Model Moments	96
20.4.1 Empirical moments	96
20.4.2 Simulated moments	96
20.4.3 Theoretical moments	96

21 Forecasting	97
21.1 Unconditional forecast	97
21.2 Conditional forecast	98
21.3 Forecast evaluation	98
21.3.1 Mincer-Zarnowitz test	98
21.3.2 Diebold-Mariano test	98
21.3.3 PIT	98
21.4 Forecast combination	98
21.5 Aggregate forecast	99
21.6 Data transformation and reporting	102
21.7 Evaluate forecast from model	103
22 Model selection	104
23 More examples	105
24 Code references	105
25 Library	107
25.1 Date classes and functions	107
25.2 Data management classes and functions	455
25.3 Overhead graphics classes and functions	1299
25.4 Basic graphics classes and functions	1598
25.5 Utils graphics classes and functions	1868
25.6 Cell functions	1869
25.7 Char functions	1880
25.8 Help Classes	1884
25.9 Double functions	1965
25.10 IO functions	2004
25.11 Struct functions	2014
25.12 Other functions	2031
25.13 Calendars	2060
25.14 Diagnostics	2107
25.15 Distributions	2189
25.16 Estimation	2442
25.17 Forecasting	2672
25.18 Models	2717
25.19 Optimization	4882
25.20 Parallel coding	4920
25.21 Solvers	4924
25.22 Utils	4978

1 Setup toolbox

You can add functions and classes of this toolbox with the `nb_startup` function. To be able to run this function you must add the main folder of the toolbox to the MATLAB paths. E.g:

```
addpath('XXX\NBTOOLBOX-main');
```

where XXX refers to the path you have placed the NBTOOLBOX-main folder. You can now execute the `nb_startup` function. It has two optional inputs. The first is the wanted functionalities:

- `'all'`: Same as `'gui'`.
- `'dg'`: Only add the data management and graphics functions and classes.
- `'full'`: Will add all the functionalities of the toolbox, except the GUI part. Default.
- `'g'`: Only add the basic plot functions and classes.
- `'gui'`: Add all the functionalities needed to get the GUI classes for graphics.

With the second input you could tell the `nb_startup` function not to print anything on the screen. I.e. give 'silent'.

1.1 System requirements

You need eps2pdf and ghostscript installed on your PC to be able to save down figures and tables to PDFs. You need MiKTeX\2.8 or similar to be able to produce LaTeX related outputs.

1.2 Resolution and screen setup

You can set the property `plotAspectRatio`, of the classes `nb_graph_ts`, `nb_graph_cs` or `nb_graph_ts` to make the graphs produced invariant to the screen resolution. I.e set it to `'[4,3]'` or `'[16,9]'`. `'[16,9]'` is default for the graph styles `'presentation'` and `'presentation_white'`. Do not dock the figures, it may result in a lot of white space around the graph in the printed files and a warning may be displayed. To prevent this, write the following on the command line:

```
set(0,'DefaultFigureWindowStyle','normal');
```

2 Theoretical foundation

See [Paulsen \(2021\)](#) for a theoretical discussion of the methods made available in this MATLAB code. I.e. check the corresponding section header.

3 Get help on classes and functions

Of course you can use this manual to look for help on some classes or functions, but sometimes it could be easier to use some MATLAB functionalities to get this help inside MATLAB. The main function is the MATLAB function `help`. Say you want to get the

some help on the class `nb_ts`, then you can type in the following on the command line; `help nb_ts`. A full description of this class will then be printed on the command line, including how to initialize an object of this class, its properties and its methods. This will show you that the `nb_ts` class has the method `abs`, but it does not say anything about what this method does or how to use it. If you want some help one this method, you can type in the following on the command line; `help nb_ts.abs`. In the same way you can get help on some specific property of the `nb_ts` class, for example the property `data`, by typing in the following on the command line; `help nb_ts.data`.

For all classes that can be initialized with no input, you can type in `methods(classname)` and `properties(classname)` to get the methods and properties of the class respectively.

To get help on some function, say the `nb_readExcel`, you can type the following on the command line; `help nb_readExcel`.

4 Dates

This toolbox supports both dates as strings and date objects. I.e. if some function or method calls for a date as an input, both date types are supported. Some exceptions exist though. For example the functions `nb_dateplus` and `nb_dateminus`. The supported frequencies are daily (also business when read from an excel spreadsheet), weekly, monthly, quarterly, semiannually and yearly.

4.1 Dates as strings

When you use/call functions or methods of this toolbox it is easiest to use date inputs as strings. The code will be easier to read. However, using date inputs as strings require a strict date format. In this toolbox the following date formats are supported:

- Daily: 'dd.mm.yyyy' or 'yyyyMm(m)D(d)'
- Weekly: 'dd.mm.yyyy' or 'yyyyWw(w)'
- Monthly: 'dd.mm.yyyy', 'ddmonyyyy' or 'yyyyMm(m)'
- Quarterly: 'dd.mm.yyyy' or 'yyyyQq'
- Semiannually: 'dd.mm.yyyy' or 'yyyySs'
- Yearly: 'dd.mm.yyyy' or 'yyyy'

4.2 Dates as objects

As already mentioned dates will generally be stored as objects instead of as strings in this toolbox. The main reason for this is to speed up the code and it makes it easier to write general code.

In this toolbox each frequency is represented by its own class:

- Daily: `nb_day`
- Weekly: `nb_week`
- Monthly: `nb_month`
- Quarterly: `nb_quarter`
- Semiannually: `nb_semiAnnual`
- Yearly: `nb_year`

All these classes are subclasses of the superclass `nb_date`. I will not go into the technicalities for why this is so, but again it is for practical reasons. All the date classes also share the same main methods, which makes them generic. Another important feature of these objects are the frequency property stored in each date object. This property is a scalar and its value will depend on the frequency of the date. The property will take the values:

- Daily: 365
- Weekly: 52
- Monthly: 12
- Quarterly: 4
- Semianually: 2
- Yearly: 1

So when a method or function asks for the frequency of the dates (or data) as a double one of the above given values must be provided. I.e. the number matching the wanted frequency of the dates (or data).

For some examples of the usage of the date classes I refer to the listing ??.

4.3 Examples

See the example script file found at \\NBTOOLBOX\\Examples\\Dates\\nb_datesExamples.m

5 Data management

To store data in a practical way is always important when using large amount of data. With this toolbox it is possible to store the data in objects. Mainly as a `nb_data`, `nb_ts` or `nb_cs` object. Where the `nb_data` stores dimensionless data, the `nb_ts` class stores time-series data, and the `nb_cs` stores cross-sectional data. The following parts explain the usage of all classes in turn.

5.1 Time series data

As already mentioned, the `nb_ts` class will be the main class for storing time-series data.³ This class supports reading data from excel. It also supports some MATLAB objects as inputs. When I say MATLAB objects I also mean basic objects as double and struct. Some user created objects such as tseries (IRIS toolbox) and `nb_math_ts` can also be "read" by the `nb_ts` class. I will now describe how to read data from the different sources into an object of class `nb_ts`. I.e. how to initialize an object.

5.1.1 Read data from excel

To import an excel spreadsheet into an `nb_ts` object in MATLAB you can write the following:

```
obj = nb_ts('example_ts_quarterly.xlsx');
```

³`nb_ts` is a value class. Therefore you must assign all method calls when using this class. I.e. `obj = obj.growth()` not `obj.growth()`

Here the data saved in the excel spreadsheet is stored in the object obj, which will be an object of class `nb_ts`. See table 1 for an example of the needed format of the excel spreadsheet. If you now write obj on the command line you will get the following:

```
obj =
```

'Time'	'Var1'	'Var2'	'Var3'
'2000Q1'	[0.6280]	[0.9831]	[0.8555]
'2000Q2'	[0.2920]	[0.3015]	[0.6448]
'2000Q3'	[0.4317]	[0.7011]	[0.3763]
'2000Q4'	[0.0155]	[0.6663]	[0.1909]
'2001Q1'	[0.9841]	[0.5391]	[0.4283]
'2001Q2'	[0.1672]	[0.6981]	[0.4820]
'2001Q3'	[0.1062]	[0.6665]	[0.1206]
'2001Q4'	[0.3724]	[0.1781]	[0.5895]
'2002Q1'	[0.1981]	[0.1280]	[0.2262]
'2002Q2'	[0.4897]	[0.9991]	[0.3846]
'2002Q3'	[0.3395]	[0.1711]	[0.5830]
'2002Q4'	[0.9516]	[0.0326]	[0.2518]
'2003Q1'	[0.9203]	[0.5612]	[0.2904]
'2003Q2'	[0.0527]	[0.8819]	[0.6171]
'2003Q3'	[0.7379]	[0.6692]	[0.2653]
'2003Q4'	[0.2691]	[0.1904]	[0.8244]
'2004Q1'	[0.4228]	[0.3689]	[0.9827]
'2004Q2'	[0.5479]	[0.4607]	[0.7302]
'2004Q3'	[0.9427]	[0.9816]	[0.3439]
'2004Q4'	[0.4177]	[0.1564]	[0.5841]

Which looks a lot like the excel spreadsheet. It is important to note that the `nb_ts` class only read the first worksheet of the excel file as default. The read worksheet must only contain the data to read (No comments or figures). The required date format is the same as described in the section 4. With the second input you can specify a specific worksheet to read. It must then be given as a string.⁴. Please see the example below.

```
obj = nb_ts('example_ts_quarterly', 'sheet1')
```

If you only want a subset of the data stored in the excel spreadsheet read the data as ascribed above, and then use the `window` method of the `nb_ts` class to shrink the data to the wanted dates and/or variables.

5.1.2 Read data from a .mat file

It is also possible to load data from a .mat file with the code:

```
obj = nb_ts('example_ts_quarterly_mat')
```

To use .mat files are generally much faster than using excel spreadsheets. Type help `nb_ts` on the command line to get more info on the needed format of the .mat file or see the example file stored in the folder:

`\NBTOOLBOX\Examples\DataManagement`.

⁴If it does not find the provided sheet name it will read the default sheet, and it will only add the provided string as the name of the data set.

	Time	Var1	Var2	Var3
2000Q1	0.6280	0.9831	0.8555	
2000Q2	0.2920	0.3015	0.6448	
2000Q3	0.4317	0.7011	0.3763	
2000Q4	0.0155	0.6663	0.1909	
2001Q1	0.9841	0.5391	0.4283	
2001Q2	0.1672	0.6981	0.4820	
2001Q3	0.1062	0.6665	0.1206	
2001Q4	0.3724	0.1781	0.5895	
2002Q1	0.1981	0.1280	0.2262	
2002Q2	0.4897	0.9991	0.3846	
2002Q3	0.3395	0.1711	0.5830	
2002Q4	0.9516	0.0326	0.2518	
2003Q1	0.9203	0.5612	0.2904	
2003Q2	0.0527	0.8819	0.6171	
2003Q3	0.7379	0.6692	0.2653	
2003Q4	0.2691	0.1904	0.8244	
2004Q1	0.4228	0.3689	0.9827	
2004Q2	0.5479	0.4607	0.7302	
2004Q3	0.9427	0.9816	0.3439	
2004Q4	0.4177	0.1564	0.5841	

Table 1: Format of the excel spreadsheet

5.1.3 Initialize an data object with MATLAB types

There is also possible to initialize an object of class `nb_ts` with different MATLAB data types as inputs. I will not go into detail on all these possibilities, but some examples will now be provided. First an example of initializing an object with a double⁵:

```
obj = nb_ts([2,2;2,2;2,2], 'double', '2012M1', {'Var1', 'Var2'})
```

The result will be (when the object is displayed on the command line):

```
obj =
    'Time'      'Var1'      'Var2'
    '2012M1'    [ 2]      [ 2]
    '2012M2'    [ 2]      [ 2]
    '2012M3'    [ 2]      [ 2]
```

It is also possible to initialize an object of class `nb_ts` with user defined objects, such as tseries objects provided by the IRIS package and `nb_math_ts` objects provided by the NB toolbox. See the section 5.1.14 for more on the `nb_math_ts` class. As both these types of objects do not have a reference to the variables stored in the objects, it will be important to provide them when calling the constructor of the `nb_ts` class. An example:

```
nb_math_tsObj = nb_math_ts([2,2;2,2;2,2], '2012M1');
obj           = nb_ts(nb_math_tsObj, '', '', {'Var1', 'Var2'})
```

The object `obj` will represent the same data as the example above. See the documentation on the `nb_ts` class found in the library for a more comprehensive description of all the different ways of providing data to an object of class `nb_ts`.

⁵The 'double' string provided here is not necessary to be able to initialize an object of class `nb_ts` with a double. It only provides the data set a name.

5.1.4 Transformation of the data

The time-series data stored in an object of class `nb_ts` can be transformed by different methods of the class. An example is the `egrowth` method which calculates the growth rate of the time-series stored in the object. It is important to note that all the variables stored in the object is transformed by this method. Calling the method of the object can be done by writing:

```
obj = nb_ts('example_ts_quarterly')
obj = obj.egrowth();
```

Now the object `obj` stores all the growth series of the same variables instead. There are many supported methods that does different transformations of the time-series stored in an `nb_ts` object. All these methods are documented in the [library](#).

5.1.5 Creating new variables

The `nb_ts` class has a special method for creating (adding) a new variable to the object. This can be done by writing:

```
obj = nb_ts('example_ts_quarterly');
obj = obj.deleteVariables('Var3');
obj = obj.createVariable('New variable', 'Var1-Var2')
```

Which will result in (when display on the command line):

```
obj =

    'Time'      'New variable'      'Var1'      'Var2'
    '2000Q1'    [-0.3551]        [0.6280]    [0.9831]
    '2000Q2'    [-0.0095]        [0.2920]    [0.3015]
    '2000Q3'    [-0.2694]        [0.4317]    [0.7011]
    '2000Q4'    [-0.6509]        [0.0155]    [0.6663]
    '2001Q1'    [0.4449]         [0.9841]    [0.5391]
    '2001Q2'    [-0.5309]        [0.1672]    [0.6981]
    '2001Q3'    [-0.5603]        [0.1062]    [0.6665]
    '2001Q4'    [0.1943]         [0.3724]    [0.1781]
    '2002Q1'    [0.0701]         [0.1981]    [0.1280]
    '2002Q2'    [-0.5094]        [0.4897]    [0.9991]
    '2002Q3'    [0.1684]         [0.3395]    [0.1711]
    '2002Q4'    [0.9190]         [0.9516]    [0.0326]
    '2003Q1'    [0.3591]         [0.9203]    [0.5612]
    '2003Q2'    [-0.8292]        [0.0527]    [0.8819]
    '2003Q3'    [0.0687]         [0.7379]    [0.6692]
    '2003Q4'    [0.0787]         [0.2691]    [0.1904]
    '2004Q1'    [0.0539]         [0.4228]    [0.3689]
    '2004Q2'    [0.0871]         [0.5479]    [0.4607]
    '2004Q3'    [-0.0389]        [0.9427]    [0.9816]
    '2004Q4'    [0.2613]         [0.4177]    [0.1564]
```

We now see that a new variable have been added. The inputs to the `createVariable` method must be the input data stored in the `nb_ts` object, a string with the name of the new variable and a string with the expression to be evaluated. In the expression to be evaluated all the names of the variables stored in the `nb_ts` object can be used. All the special mathematical operator (+, -, .*, .^, ./) and methods of the `nb_math_ts` class are

supported.⁶ See section 5.1.14 or the [library](#) for more on the `nb_math_ts` class. It is also possible to create more new variables at the same time with one method call. Then the inputs to the method must be cell arrays of strings (`cellstr`) with the same length. For example:

```
obj = nb_ts('example_ts_quarterly');
obj = obj.createVariable({'Var4','Var5'},...
{'Var1-Var2','Var1./Var2'})
```

5.1.6 Converting data to a new frequency

At times a method for converting data to a different frequency is wanted. This could be done by the `convert` method of the `nb_ts` class. We will first start with an example:

```
obj = nb_ts('example_ts_quarterly');
obj = obj.convert(1)
```

Which will result in (when display on the command line):

```
obj =
    'Time'      'Var1'      'Var2'      'Var3'
    '2000'      [0.0155]   [0.6663]   [0.1909]
    '2001'      [0.3724]   [0.1781]   [0.5895]
    '2002'      [0.9516]   [0.0326]   [0.2518]
    '2003'      [0.2691]   [0.1904]   [0.8244]
    '2004'      [0.4177]   [0.1564]   [0.5841]
```

Here in this example we converted the frequency of the data from quarterly (4) to yearly (1). When calling the `convert` input with only one input (besides the `nb_ts` object itself) it must be [the frequency to be converted to as a double](#). If no further input are given it is default to take the last observation in each subperiod when converting the data to a lower frequency and to set the first period of the subperiods to the values of the lower frequency when converting to a higher frequency, where all the data points in between are set to nan. The following example converts the data from quarterly to monthly:

```
obj = nb_ts('example_ts_quarterly');
obj = obj.window('2000Q1','2000Q4');
obj = obj.convert(12)
```

Which will result in (when display on the command line):

```
obj =
    'Time'      'Var1'      'Var2'      'Var3'
    '2000M1'    [0.6280]   [0.9831]   [0.8555]
    '2000M2'    [  NaN]     [  NaN]     [  NaN]
    '2000M3'    [  NaN]     [  NaN]     [  NaN]
    '2000M4'    [0.2920]   [0.3015]   [0.6448]
    '2000M5'    [  NaN]     [  NaN]     [  NaN]
    '2000M6'    [  NaN]     [  NaN]     [  NaN]
    '2000M7'    [0.4317]   [0.7011]   [0.3763]
    '2000M8'    [  NaN]     [  NaN]     [  NaN]
```

⁶ Actually when calling the `createVariable` method all variables are dumped to `nb_math_ts` objects, which makes all methods defined for this class work (including all the special mathematical operators defined in MATLAB).

```
'2000M9'      [   NaN]      [   NaN]      [   NaN]
'2000M10'     [0.0155]    [0.6663]    [0.1909]
```

If you want to convert the data in another way there exists some more possibilities. When converting to a lower frequency the following methods for converting are supported:

- 'average': Takes averages over the subperiods
- 'discrete': Takes last observation in each subperiod (Default)
- 'first': Takes first observation in each subperiod
- 'max': Takes max observation in each subperiod
- 'min': Takes min observation in each subperiod
- 'sum': Takes sums over the subperiods

When converting to a higher frequency the following options are supported⁷:

- 'cubic': Shape-preserving piecewise cubic interpolation
- 'daverage': Uses the Denton 1971 method using that the average of the intra low frequency periods should preserve the average.
- 'dsum': Uses the Denton 1971 method using that the sum of the intra low frequency periods should preserve the sum.
- 'fill': No interpolation. All periods of the higher frequency get the same value as that of the lower frequency.
- 'linear': Linear interpolation
- 'none': No interpolation. All data in between is given by nans (missing values)(Default)
- 'spline': Piecewise cubic spline interpolation

So for example if we want to convert the quarterly data given in the last example by the linear interpolation we can write the following:

```
obj = nb_ts('example_ts_quarterly');
obj = obj.window('2000Q1','2000Q4');
obj = obj.convert(12,'linear')
```

Which will result in (when display on the command line):

```
obj =

```

'Time'	'Var1'	'Var2'	'Var3'
--------	--------	--------	--------

⁷All these option will by default use the first period of the subperiods as base for the conversion. Provide the optional input 'interpolateDate' if you want to set the base to the end periods instead ('end').

```

'2000M1'      [0.6280]    [0.9831]    [0.8555]
'2000M2'      [0.5160]    [0.7559]    [0.7853]
'2000M3'      [0.4040]    [0.5287]    [0.7150]
'2000M4'      [0.2920]    [0.3015]    [0.6448]
'2000M5'      [0.3385]    [0.4347]    [0.5553]
'2000M6'      [0.3851]    [0.5679]    [0.4658]
'2000M7'      [0.4317]    [0.7011]    [0.3763]
'2000M8'      [0.2929]    [0.6895]    [0.3145]
'2000M9'      [0.1542]    [0.6779]    [0.2527]
'2000M10'     [0.0155]    [0.6663]    [0.1909]

```

If it is wanted to change the base for the interpolation when converting, it could be done as follows. (Be aware that you must provide the second option when providing any of the optional options. I.e. the string with the conversion method.):

```

obj = nb_ts('example_ts_quarterly');
obj = obj.window('2000Q1','2000Q4');
obj = obj.convert(12,'linear','interpolateDate','end')

```

Which will result in (when display on the command line):

```

obj =

'Time'      'Var1'      'Var2'      'Var3'
'2000M3'    [0.6280]    [0.9831]    [0.8555]
'2000M4'    [0.5160]    [0.7559]    [0.7853]
'2000M5'    [0.4040]    [0.5287]    [0.7150]
'2000M6'    [0.2920]    [0.3015]    [0.6448]
'2000M7'    [0.3385]    [0.4347]    [0.5553]
'2000M8'    [0.3851]    [0.5679]    [0.4658]
'2000M9'    [0.4317]    [0.7011]    [0.3763]
'2000M10'   [0.2929]    [0.6895]    [0.3145]
'2000M11'   [0.1542]    [0.6779]    [0.2527]
'2000M12'   [0.0155]    [0.6663]    [0.1909]

```

5.1.7 Merging data

Another important method of the `nb_ts` class is the `merge` method. This method makes it possible to merge two `nb_ts` objects into one `nb_ts` object storing all the variables from the two input objects. An example will follow:

```

obj1 = nb_ts(ones(4,2),'', '2012', {'Var1','Var2'});
obj2 = nb_ts(ones(4,1),'', '2012', 'Var3');
obj3 = nb_ts(ones(4,2),'', '2012', {'Var1','Var3'});

% Merging two objects
obj4 = obj1.merge(obj2)
obj5 = obj1.merge(obj3)

```

Both will result in an object representing the same data, as is seen when display on the command line:

```

obj4 =

'Time'      'Var1'      'Var2'      'Var3'
'2012'      [ 1]       [ 1]       [ 1]
'2013'      [ 1]       [ 1]       [ 1]
'2014'      [ 1]       [ 1]       [ 1]

```

```

'2015'      [ 1]      [ 1]      [ 1]

obj5 =
    'Time'      'Var1'      'Var2'      'Var3'
    '2012'      [ 1]      [ 1]      [ 1]
    '2013'      [ 1]      [ 1]      [ 1]
    '2014'      [ 1]      [ 1]      [ 1]
    '2015'      [ 1]      [ 1]      [ 1]

```

This example shows that it is still possible to merge two objects storing the the same variable. This will be the case only as long as the variable represent the same time-series. It is also possible to merge databases with different frequencies. E.g:

```

obj1 = nb_ts(ones(2,2),'', '2012', {'Var1','Var2'});
obj2 = nb_ts(ones(8,1),'', '2012Q1', 'Var3');
obj3 = obj1.merge(obj2)

```

Will result in (when display on the command line):

```

obj3 =
    'Time'      'QUA_Var1'      'QUA_Var2'      'QUA_Var3'
    '2012Q1'      [ 1]      [ 1]      [ 1]
    '2012Q2'      [ NaN]      [ NaN]      [ 1]
    '2012Q3'      [ NaN]      [ NaN]      [ 1]
    '2012Q4'      [ NaN]      [ NaN]      [ 1]
    '2013Q1'      [ 1]      [ 1]      [ 1]
    '2013Q2'      [ NaN]      [ NaN]      [ 1]
    '2013Q3'      [ NaN]      [ NaN]      [ 1]
    '2013Q4'      [ NaN]      [ NaN]      [ 1]

```

As you see the data with the lowest frequency will be converted with the use of the `convert` method of the `nb_ts` class (with default inputs). If you want to change the way the data with the lowest frequency is converted you can provide more inputs to the `merge` method. See the examples that follows for more on these inputs:

```

% The default setting for merging data with different frequencies
obj3 = obj1.merge(obj2, 'start', 'none', 'on')

% (Only) change the base for the merging (Same converting method
% and with renaming)
obj3 = obj1.merge(obj2, 'end')

% Do linear interpolation with renaming
obj3 = obj1.merge(obj2, 'start', 'linear')

% Do linear interpolation without renaming
obj3 = obj1.merge(obj2, 'start', 'linear', 'off')

```

See the `library` for more on the `merge` method.

5.1.8 Writing data to Excel

If you want to write the data to a excel spreadsheet you can write:

```

obj = nb_ts(ones(8,2), 'filename', '2012Q1', {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase()

```

In this example you will write the data to a file, where the name of that file will be given by the name of the data set. In this example 'filename'. (This name is stored in the property `dataNames` of the `nb_ts` object).

If you want to save the file to another name you can use the 'saveName' option. I.e:

```
obj = nb_ts(ones(8,2), 'filename', '2012Q1', {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase('saveName', 'test')
```

It is also possible to change the date format of the written excel spreadsheet:

```
obj = nb_ts(ones(8,2), 'filename', '2012Q1', {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase('saveName', 'test', 'dateFormat', 'xls')
```

The date formats supported are:

- 'default'
- 'fame'
- 'xls'
- 'NBNorsk' ('NBNorwegian')
- 'NBEnglish' ('NBEngelsk')

See the method `toDates` of the date classes for more on these different date formats. (I.e. `nb_day`, `nb_week`, `nb_month`, `nb_quarter`, `nb_semiAnnual` and `nb_year`.)

If your object consists of more then one data set (have more pages) the default will be to save each data set as a separate excel spreadsheet. (The default will again be to use the names of the data sets stored in the `dataNames` property of the object as the names of the saved files.) If you instead want to write all the data sets of the object to one excel file you can write (Each data sets will be stored as separate worksheets):

```
obj = nb_ts(ones(8,2,2), {'data1', 'data2'}, '2012Q1',...
            {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase('saveName', 'test', 'append', 1)
```

If the data are saved in this way it is not possible to read all the worksheets of the excel spreadsheet to an `nb_ts` object again. But you can use the `nb_readExcelMoreSheets` function to read it to an `nb_DataCollection` object. For more information on the `saveDataBase` look it up in the [library](#).

5.1.9 Writing data to other file formats

The method `saveDataBase` makes it also possible to write an `nb_ts` object to a .mat file or .txt file. This can be done with the 'ext' option:

```
% Save the data to a .mat file
obj = nb_ts(ones(8,2), 'filename', '2012Q1', {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase('saveName', 'filename', 'ext', 'mat')

% Save the data to a .txt file
obj = nb_ts(ones(8,2), 'filename', '2012Q1', {'QUA_Var1', 'QUA_Var2'});
obj.saveDataBase('ext', 'txt', 'saveName', 'filename')
```

From the example you can see that it is possible to save data of the object to a .txt ('txt') and .mat ('mat'), these are the only file formats supported besides excel ('xlsx').

5.1.10 Converting to other data types (objects)

In case you want to transform the `nb_ts` object to any other MATLAB object/type. The `nb_ts` class has some methods for doing this. See the examples below for how to do this.

```
data = nb_ts([2,2,2;2,2,2], '', '1994Q1', {'Var1', 'Var2', 'Var3'});  
  
% Convert to a structure  
s = data.toStructure()  
  
% Convert to a cell  
c = data.asCell()  
  
% Convert to a double  
d = data.double()  
  
% Convert to a nb_math_ts object  
mts = nb_math_ts(data)  
  
% Convert to a nb_cs object  
cs = data.tonb_cs();  
  
% Convert to a nb_data object  
data = data.tonb_data();
```

5.1.11 Mathematical operators

Mathematical operators act on `nb_ts` objects in a special way. The operators `.*`, `.^`, `+`, `-` and `./` will try to match the variables found in both objects and do the calculation on these variables only. The rest of the variables will be given nan values for all data points. So for example if an `nb_ts` object which includes the variables `var1` and `var2` and another object includes only `var1`, then the operator will only act on the data of the variable `var1` of the returned `nb_ts` object while the `var2` will have all nan values. When the dates of the two objects does not match all the data points outside the window of both object will also be returned as nan values.

The operators `*`, `^`, `+`, `-` and `/` act on all the data of the object, but they are only defined when combined with scalars. See the examples below.

```
data1 = nb_ts(ones(2,3)*3, '', '1994Q1', {'Var1', 'Var2', 'Var3'});  
data2 = nb_ts(ones(2,3)*2, '', '1994Q1', {'Var1', 'Var2', 'Var3'});  
  
% Element-wise operators  
data = data1-data2      % Element-wise minus  
data = data1+data2      % Element-wise plus  
data = data1.*data2     % Element-wise multiplication  
data = data1.^data2     % Element-wise power  
data = data1./data2     % Element-wise division  
  
% Operators that act on all the data of the object  
data = data1-2          % Minus (Only for scalars)  
data = data1+2          % Plus (Only for scalars)  
data = data1*2          % Multiplication (Only defined when  
                      % multiplied with a scalar)  
data = data1^2          % Power (Only defined when taken power with  
                      % a scalar)  
data = data1/2          % Only defined when divided by a scalar
```

5.1.12 Logical operators

The `nb_ts` also supports the logical operators `|`, `>`, `>=`, `==`, `<=`, `<` & `~`. All these operators will try to match the variables found in the objects and do the test element-wise

on the data of the matching variables. Contrary to the mathematical operators these operators will not work as long as the two objects do not include the same variables and dates. Examples are found below.

```

data1 = nb_ts(logical([ones(2,2), ...
    zeros(2,1)]),'', '1994Q1', {'Var1', 'Var2', 'Var3'});
data2 = nb_ts(logical([ones(2,1), ...
    zeros(2,2)]),'', '1994Q1', {'Var1', 'Var2', 'Var3'});

data = data1 & data2
data = data1 | data2

data1 = nb_ts(ones(2,3)*3, '', '1994Q1', {'Var1', 'Var2', 'Var3'});
data2 = nb_ts(ones(2,3)*2, '', '1994Q1', {'Var1', 'Var2', 'Var3'});

data = data1 > data2
data = data1 >= data2
data = data1 == data2
data = data1 <= data2
data = data1 < data2
data = data1 ~= data2

```

5.1.13 Other useful methods

There are also many other methods of the class which could make programming with time-series data more easy. I will not describe these methods here, but instead provide a list with links to the library at the end of this reference manual.

- [addVariable](#) Add a new variable to the object
- [dates](#) Return the dates of the object as a cellstr
- [deleteVariables](#) Delete some variables of the object
- [getRealEndDate](#) Get the last observation which is not nan or infinite
- [getRealStartDate](#) Get the first observation which is not nan or infinite
- [isempty](#) Checks if an object is empty
- [keepVariables](#) Keep some variables of the object
- [rename](#) Rename some variables or types of the object
- [setValue](#) Set the value of some data of the object
- [structure2Dataset](#) Add a data set from a structure
- [window](#) Shrink the window of the object

Type

```
methods(nb_ts)
```

in the command line to get the full list of method for the `nb_ts` class.

5.1.14 The `nb_math_ts` class

As mentioned earlier, the `nb_ts` class stores the data with reference to the variable names. This can be useful in many settings. For example when doing data manipulation as you can do with doubles, but at the same time have the data matched with some dates. If

so the `nb_math_ts` class could be utilized. This class has many of the same methods as the `nb_ts` class, but of course the input will differ given that the data has no link to the variable names.

The big difference is the mathematical operators `.*`, `.^`, `+`, `-` and `./`. They will now only be defined for object with the same number of observations (and same dates), and pages. In addition the number of variables (columns) of the object must either match, or one of the objects must have only one column. The examples below will show how these operators could be utilized to transform data.

```
%% Initializing an nb_math_ts object

var1 = nb_math_ts([1;2;3;4;5], '1994Q1')
var2 = nb_math_ts([0;1;2;3;4], '1994Q1')

%% Mathematical operators

var3 = var1-var2
var4 = var3./var1
var5 = (var4.*var2)./var3
```

When using the `createVariable` method of the `nb_ts` class, the functionality of the `nb_math_ts` class is being utilized. As these functionalities are very similar to the functionalities of double you can just pretend that the data are doubles. But contrary to the double class created by MATLAB, some special functions which relates to dates are defined for the `nb_math_ts` class:

- `reIndex` Makes it possible to re-indexing a time-series to a new date
- `sumOAF` Take the sum over a lower frequency

5.1.15 Examples

See the example script file found at \\NBTOOLBOX\Examples\DataManagement\nb_tsExamples.m

5.2 Cross sectional data

The `nb_cs` class is the main class for storing case data (cross-sectional data).⁸ You can read data from a excel spreadsheet and .mat (With a specific format). It is also possible to initialize an object of class `nb_cs` with MATLAB types as double and struct. For more see the following subsections. This class stores the data in a specific way. Where we from now on will call the first dimension the types dimension and the second dimension the variables dimension. The types of the data will be stored in the property `types` and the variables will be stored in the property `variables` of the `nb_cs` object. Both as cells with strings (cellstr).

5.2.1 Read data from excel

To read data from excel is simple. Just provide the name of the excel spreadsheet as the first input (with or without the extension). E.g:

```
obj = nb_cs('example_cs.xlsx')
```

⁸ `nb_cs` is a value class. Therefore you must assign all method calls when using this class. I.e. `obj = obj.growth()` not `obj.growth()`

Here the the data saved in the excel spreadsheet is stored in the object obj, which will be an object of class `nb_cs`. See table 2 for an example of the needed format of the excel spreadsheet. So if you now write obj on the command line you will get the following:

```
obj =
'Types'      'Var1'      'Var2'      'Var3'
'type1'      [ 2]      [ 2]      [ 2]
'type2'      [ 2]      [ 2]      [ 2]
'type3'      [ 2]      [ 2]      [ 2]
```

Which looks very much the same as the excel spreadsheet. It is important to note that the `nb_cs` class only reads the first worksheet of the excel file by default. The read worksheet must only contain the data to read (No comments or figures). By the second input you can specify a specific worksheet to read. It must then be given as a string.⁹

Types	Var1	Var2	Var3
type1	2	2	2
type2	2	2	2
type3	2	2	2

Table 2: Format of the excel spreadsheet

It is also possible to read a specific worksheet of the excel spreadsheet. Here is an example reading the worksheet sheet1.

```
obj = nb_cs('example_cs.xlsx','sheet1')
```

If you only want a subset of the data stored in the excel spreadsheet read the data as ascribed above, and then use the `window` method of the `nb_cs` class to shrink the data to the wanted types and/or variables.

5.2.2 Read data from a .mat file

It is also possible to load data from a .mat file with the code:

```
obj = nb_cs('example_cs_mat.mat')
```

To use .mat files are generally much faster then using excel spreadsheets. Type help `nb_cs` on the command line to get more info on the needed format of the .mat file or see the example file stored in the folder:

`\NBTOOLBOX\Examples\DataManagement.`

5.2.3 Initialize an data object with MATLAB types

It is also possible to initialize an object of class `nb_cs` with MATLAB data types as inputs. I will not go into detail on all these possibilities, but I will provide an example on how to initialize an object of class `nb_cs` with a double¹⁰:

```
obj = nb_cs([2,2,2;2,2,2;2,2,2], 'double',...
{'type1','type2','type3'}, {'Var1','Var2','Var3'})
```

⁹If it does not find the provided sheet name it will read the default sheet, and it will only add the provided string as the name of the data set.

¹⁰The 'double' string provided here is not necessary to be able to initialize an object of class `nb_ts` with a double. It only provides the data set a name.

Here we have that the first input is the double matrix, the second input will be the name of the data provided, the third input will be the types of the data and the last input will be the names of the variables. (This will result in the same data as reading the excel example file given above.)

5.2.4 Transformation of the data

The case data stored in an object of class `nb_cs` can be transform with the use of different methods of the class. An example is the `log` method which take log of the data stored in the object. It is important to note that it does this with all the variables stored in the object. Calling the method of the object can be done as follows:

```
obj = nb_cs('example_cs_mat');
obj = obj.log();
% Or
obj = log(obj);
```

Now the object `obj` stores all the variables data in logs instead. There are some supported methods which can do different transformations of the case data stored in an `nb_cs` object. All these methods are documented in the [library](#).

5.2.5 Creating new variables and types

The `nb_cs` class has a special method for creating (adding) a new variable to the object. This could be done as follows:

```
obj = nb_cs('example_cs');
obj = obj.createVariable('Var4', 'Var1./Var2')
```

Which will result in (when display on the command line):

```
obj =
    'Types'      'Var1'      'Var2'      'Var3'      'Var4'
    'type1'      [ 2]       [ 2]       [ 2]       [ 1]
    'type2'      [ 2]       [ 2]       [ 2]       [ 1]
    'type3'      [ 2]       [ 2]       [ 2]       [ 1]
```

We now see that a new variable have been added. The inputs to the `createVariable` method must be the input data stored in the `nb_cs` object, a string with the name of the new variable and a string with the expression to be evaluated. In the expression to be evaluated all the names of the variables stored in the `nb_cs` object can be used. All the special mathematical operator (+, -, .*, .^, ./) are supported.¹¹ It is also possible to create more variables at the same time with one method call. Then the inputs to the method must be cell arrays of strings (cellstr) with the same length. For example:

```
obj = nb_cs('example_cs');
obj = obj.createVariable({'Var4', 'Var0'}, {'Var1./Var2', '2*Var1'})
```

It is equally possible to create new types and add them to the object. Use the `createType` method of the `nb_cs` class.

¹¹ Actually when calling the `createVariable` method all variables are dumped to double vectors, which makes all methods defined for doubles work (including all the special mathematical operators defined in MATLAB).

5.2.6 Merging data

To merge data of different `nb_cs` objects you can utilize the `merge` method of the `nb_cs` class. E.g:

```
obj1 = nb_cs([2,2,2;2,2,2;2,2,2], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var1', 'Var2', 'Var3'});
obj2 = nb_cs([2,2,2;2,2,2;2,2,2], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var4', 'Var5', 'Var6'});
m    = merge(obj1,obj2)
```

It is also possible to merge two `nb_cs` objects which share some variables, but only as long as they also share the same data. I.e. this code works:

```
obj1 = nb_cs([2,2,2;2,2,2;2,2,2], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var1', 'Var2', 'Var3'});
obj2 = nb_cs([2,2,2;2,2,2;2,2,2], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var3', 'Var4', 'Var5'});
m    = merge(obj1,obj2)
```

But not this code (because the data of the same variable differ):

```
obj1 = nb_cs([2,2,2;2,2,2;2,2,1], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var1', 'Var2', 'Var3'});
obj2 = nb_cs([2,2,2;2,2,2;2,2,2], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var3', 'Var4', 'Var5'});
m    = merge(obj1,obj2)
```

It is also possible to merge `nb_cs` objects which does not share the same types. And it is also possible to merge two objects which has the same type(s) as long as they represents the same data as the following example will illustrate:

```
obj1 = nb_cs([2,2,2;2,2,2;1,1,1], 'double', ...
             {'type1', 'type3', 'type2'}, {'Var1', 'Var2', 'Var3'});
obj2 = nb_cs([1,1,1;2,2,2], 'double', ...
             {'type2', 'type5'}, {'Var1', 'Var2', 'Var3'});
m    = merge(obj1,obj2)
% or
m = [obj1;obj2]
```

5.2.7 Writing data to Excel

When you want to write the data to a excel spreadsheet. You can write:

```
obj = nb_cs([2,2,2;2,2,2;1,1,1], 'double', ...
            {'type1', 'type3', 'type2'}, {'Var1', 'Var2', 'Var3'});
obj.saveDataBase()
```

In this example you will write the data to a file, where the name of that file will be given by the name of the data set. In this example 'double'. (This name is stored in the property `dataNames` of the `nb_cs` object).

If you want to save the file to another name you can use the 'saveName' option. I.e:

```

obj = nb_cs([2,2,2;2,2,2;1,1,1], 'double',...
            {'type1','type3','type2'}, {'Var1','Var2','Var3'});
obj.saveDataBase('saveName', 'test')

```

If your object consists of more then one data set (has more pages) the default will be to save each data set as a separate excel spreadsheet. (The default will again be to use the names of the data sets stored in the dataNames property of the object as the names of the saved files.) If you instead want to write all the data sets of the object to one excel file you can write (Each data set will be stored as a separate worksheet):

```

obj = nb_cs(ones(3,3,2)*2, 'double',...
            {'type1','type3','type2'}, {'Var1','Var2','Var3'});
obj.saveDataBase('saveName', 'test', 'append', 1)

```

If the data are saved in this way it will not be possible to read all the worksheets of the excel spreadsheet to an `nb_cs` object again. But you can use the `nb_readExcelMoreSheets` function to read it to a `nb_DataCollection` object. For more information on the `saveDataBase` look it up in the [library](#).

5.2.8 Writing data to other file formats

The method `saveDataBase` makes it also possible to write an `nb_ts` object to a .mat file or .txt file. This can be done with the 'ext' option:

```

% Save the data to a .mat file
obj = nb_cs([2,2,2;2,2,2;1,1,1], 'double',...
            {'type1','type3','type2'}, {'Var1','Var2','Var3'});
obj.saveDataBase('saveName', 'filename', 'ext', 'mat')

% Save the data to a .txt file
obj = nb_cs([2,2,2;2,2,2;1,1,1], 'double',...
            {'type1','type3','type2'}, {'Var1','Var2','Var3'});
obj.saveDataBase('ext', 'txt', 'saveName', 'filename')

```

From the example you can see that it is possible to save data of the object to a .txt ('txt') and .mat ('mat'), these are the only file formats supported besides excel ('xlsx').

5.2.9 Converting to other data types (objects)

In case you want to transform the `nb_cs` object to any other MATLAB object/type. The `nb_cs` class has some methods for doing this. See the examples below for how to do this.

```

data = nb_cs([2,2,2;2,2,2;1,1,1], 'double',...
            {'type1','type3','type2'}, {'Var1','Var2','Var3'});

% Convert to a structure
s = data.toStructure()

% Convert to a cell
c = data.asCell()

% Convert to a double
d = data.double()

```

5.2.10 Mathematical operators

The mathematical operators work in the same way as is the case for the `nb_ts` class, see section [5.1.11](#), except that now each row is linked to a type instead of a date.

5.2.11 Logical operators

The logical operators work in the same way as is the case for the `nb_ts` class, see section 5.1.12, except that now each row is linked to a type instead of a date.

5.2.12 Other useful methods

There are also many other methods of the class which could make programming with case data easier. I will not describe these methods here, but instead provide a list with links to the library at the end of this reference manual.

- `addType` Add a new type to the object
- `addVariable` Add a new variable to the object
- `deleteVariables` Delete some variables of the object
- `isempty` Checks if an object is empty
- `keepVariables` Keep some variables of the object
- `rename` Rename some variables or types of the object
- `setValue` Set the value of some data of the object
- `structure2Dataset` Add a data set from a structure
- `window` Shrink the window of the object

5.2.13 Examples

See the example script file found at \\NBTOOLBOX\\Examples\\DataManagement\\nb_csExamples.m

5.3 Dimensionless data

The `nb_data` class will be the main class for storing data without reference to dates or types, i.e. dimensionless.¹² This class supports reading data from excel. It also supports some MATLAB objects as inputs. When I say MATLAB objects I also mean basic objects as double and struct. I will now describe how to read data from the different sources into an object of class `nb_data`. I.e. how to initialize an object.

5.3.1 Read data from excel

To import an excel spreadsheet into an `nb_data` object in MATLAB you can write the following:

```
obj = nb_data('example_data.xlsx')
```

Here the data saved in the excel spreadsheet is stored in the object `obj`, which will be an object of class `nb_data`. See table 3 for an example of the needed format of the excel spreadsheet. If you now write `obj` on the command line you will get the following:

```
obj =
    'Observation'    'Var1'      'Var2'      'Var3'
```

¹²`nb_data` is a value class. Therefore you must assign all method calls when using this class. I.e. `obj = obj.growth()` not `obj.growth()`

'1'	[0.5870]	[0.4087]	[0.4588]
'2'	[0.2077]	[0.5949]	[0.9631]
'3'	[0.3012]	[0.2622]	[0.5468]
'4'	[0.4709]	[0.6028]	[0.5211]
'5'	[0.2305]	[0.7112]	[0.2316]
'6'	[0.8443]	[0.2217]	[0.4889]
'7'	[0.1948]	[0.1174]	[0.6241]
'8'	[0.2259]	[0.2967]	[0.6791]
'9'	[0.1707]	[0.3188]	[0.3955]
'10'	[0.2277]	[0.4242]	[0.3674]
'11'	[0.4357]	[0.5079]	[0.9880]
'12'	[0.3111]	[0.0855]	[0.0377]
'13'	[0.9234]	[0.2625]	[0.8852]
'14'	[0.4302]	[0.8010]	[0.9133]
'15'	[0.1848]	[0.0292]	[0.7962]
'16'	[0.9049]	[0.9289]	[0.0987]
'17'	[0.9797]	[0.7303]	[0.2619]
'18'	[0.4389]	[0.4886]	[0.3354]
'19'	[0.1111]	[0.5785]	[0.6797]
'20'	[0.2581]	[0.2373]	[0.1366]

Which looks a lot like the excel spreadsheet. It is important to note that the `nb_data` class only read the first worksheet of the excel file as default. The read worksheet must only contain the data to read (No comments or figures). With the second input you can specify a specific worksheet to read. It must then be given as a string.¹³ Please see the example below.

Obs	Var1	Var2	Var3
1	0.5870	0.4087	0.4588
2	0.2077	0.5949	0.9631
3	0.3012	0.2622	0.5468
4	0.4709	0.6028	0.5211
5	0.2305	0.7112	0.2316
6	0.8443	0.2217	0.4889
7	0.1948	0.1174	0.6241
8	0.2259	0.2967	0.6791
9	0.1707	0.3188	0.3955
10	0.2277	0.4242	0.3674
11	0.4357	0.5079	0.9880
12	0.3111	0.0855	0.0377
13	0.9234	0.2625	0.8852
14	0.4302	0.8010	0.9133
15	0.1848	0.0292	0.7962
16	0.9049	0.9289	0.0987
17	0.9797	0.7303	0.2619
18	0.4389	0.4886	0.3354
19	0.1111	0.5785	0.6797
20	0.2581	0.2373	0.1366

Table 3: Format of the excel spreadsheet

```
obj = nb_ts('example_ts_quarterly', 'sheet1')
```

¹³If it does not find the provided sheet name it will read the default sheet, and it will only add the provided string as the name of the data set.

If you only want a subset of the data stored in the excel spreadsheet read the data as ascribed above, and then use the `window` method of the `nb_data` class to shrink the data to the wanted dates and/or variables.

5.3.2 Read data from a .mat file

It is also possible to load data from a .mat file with the code:

```
obj = nb_data('example_data_mat.mat')
```

To use .mat files are generally much faster than using excel spreadsheets. Type `help nb_data` on the command line to get more info on the needed format of the .mat file or see the example file stored in the folder:

`\NBTOOLBOX\Examples\DataManagement`.

5.3.3 Initialize an data object with MATLAB types

There is also possible to initialize an object of class `nb_data` with different MATLAB data types as inputs. I will not go into detail on all these possibilities, but some examples will now be provided. First an example of initializing an object with a double¹⁴:

```
obj = nb_data([2,2;2,2;2,2], 'double', '1', {'Var1', 'Var2'})
```

The result will be (when the object is displayed on the command line):

```
obj =  
  
'Observation'    'Var1'      'Var2'  
'1'              [ 2]        [ 2]  
'2'              [ 2]        [ 2]  
'3'              [ 2]        [ 2]
```

See the documentation on the `nb_data` class found in the library for a more comprehensive description of all the different ways of providing data to an object of class `nb_data`.

5.3.4 Transformation of the data

The data stored in an object of class `nb_data` can be transformed by different methods of the class. An example is the `egrowth` method which calculates the growth rate of the series stored in the object. It is important to note that all the variables stored in the object is transformed by this method. Calling the method of the object can be done by writing:

```
obj = nb_data('example_data')  
obj = obj.egrowth();
```

Now the object `obj` stores all the growth series of the same variables instead. There are many supported methods that does different transformations of the series stored in an `nb_data` object. All these methods are documented in the [library](#).

¹⁴The 'double' string provided here is not necessary to be able to initialize an object of class `nb_data` with a double. It only provides the data set a name.

5.3.5 Creating new variables

The `nb_data` class has a special method for creating (adding) a new variable to the object. This can be done by writing:

```
obj = nb_data('example_data');
obj = obj.deleteVariables('Var3');
obj = obj.createVariable('New variable', 'Var1-Var2')
```

Which will result in (when display on the command line):

```
obj =

    'Observation'    'New variable'    'Var1'      'Var2'
    '1'              [ 0.1783]        [0.5870]    [0.4087]
    '2'              [-0.3872]        [0.2077]    [0.5949]
    '3'              [ 0.0390]        [0.3012]    [0.2622]
    '4'              [-0.1319]        [0.4709]    [0.6028]
    '5'              [-0.4807]        [0.2305]    [0.7112]
    '6'              [ 0.6226]        [0.8443]    [0.2217]
    '7'              [ 0.0773]        [0.1948]    [0.1174]
    '8'              [-0.0708]        [0.2259]    [0.2967]
    '9'              [-0.1481]        [0.1707]    [0.3188]
    '10'             [-0.1965]        [0.2277]    [0.4242]
    '11'             [-0.0722]        [0.4357]    [0.5079]
    '12'             [ 0.2256]        [0.3111]    [0.0855]
    '13'             [ 0.6609]        [0.9234]    [0.2625]
    '14'             [-0.3708]        [0.4302]    [0.8010]
    '15'             [ 0.1556]        [0.1848]    [0.0292]
    '16'             [-0.0240]        [0.9049]    [0.9289]
    '17'             [ 0.2494]        [0.9797]    [0.7303]
    '18'             [-0.0497]        [0.4389]    [0.4886]
    '19'             [-0.4674]        [0.1111]    [0.5785]
    '20'             [ 0.0208]        [0.2581]    [0.2373]
```

We now see that a new variable have been added. The inputs to the `createVariable` method must be the input data stored in the `nb_data` object, a string with the name of the new variable and a string with the expression to be evaluated. In the expression to be evaluated all the names of the variables stored in the `nb_data` object can be used. All the special mathematical operator (+, -, .*, .^, ./) and methods of the double class are supported.¹⁵ It is also possible to create more new variables at the same time with one method call. Then the inputs to the method must be cell arrays of strings (cellstr) with the same length. For example:

```
obj = nb_data('example_data');
obj = obj.createVariable({'Var4', 'Var5'}, ...
    {'Var1-Var2', 'Var1./Var2'})
```

5.3.6 Merging data

Another important method of the `nb_data` class is the `merge` method. This method makes it possible to merge two `nb_data` objects into one `nb_data` object storing all the variables from the two input objects. An example will follow:

¹⁵Actually when calling the `createVariable` method all variables are dumped to double objects, which makes all methods defined for this class work (including all the special mathematical operators defined in MATLAB).

```

obj1 = nb_data(ones(4,2),'',1,[ 'Var1','Var2']);
obj2 = nb_data(ones(4,1),'',1,[ 'Var3']);
obj3 = nb_data(ones(4,2),'',1,[ 'Var1','Var3']));

% Merging two object
obj4 = obj1.merge(obj2)
obj5 = obj1.merge(obj3)

```

Both will result in an object representing the same data, as is seen when display on the command line:

```

obj4 =

```

'Observation'	'Var1'	'Var2'	'Var3'
'1'	[1]	[1]	[1]
'2'	[1]	[1]	[1]
'3'	[1]	[1]	[1]
'4'	[1]	[1]	[1]


```

obj5 =

```

'Observation'	'Var1'	'Var2'	'Var3'
'1'	[1]	[1]	[1]
'2'	[1]	[1]	[1]
'3'	[1]	[1]	[1]
'4'	[1]	[1]	[1]

This example shows that it is still possible to merge two objects storing the the same variable. This will be the case only as long as the variable represent the same series. See the [library](#) for more on the `merge` method.

5.3.7 Writing data to Excel

If you want to write the data to a excel spreadsheet you can write:

```

obj = nb_data(ones(8,2),'filename','1',[ 'Var1','Var2']);
obj.saveDataBase()

```

In this example you will write the data to a file, where the name of that file will be given by the name of the data set. In this example 'filename'. (This name is stored in the property `dataNames` of the `nb_data` object).

If you want to save the file to another name you can use the 'saveName' option. I.e:

```

obj = nb_data(ones(8,2),'filename','1',[ 'Var1','Var2']);
obj.saveDataBase('saveName','test')

```

If your object consists of more then one data set (have more pages) the default will be to save each data set as a separate excel spreadsheet. (The default will again be to use the names of the data sets stored in the `dataNames` property of the object as the names of the saved files.) If you instead want to write all the data sets of the object to one excel file you can write (Each data sets will be stored as separate worksheets):

```

obj = nb_data(ones(8,2,2),{'data1','data2'},'1',...
              {'Var1','Var2'});
obj.saveDataBase('saveName','test','append',1)

```

If the data are saved in this way it is not possible to read all the worksheets of the excel spreadsheet to an `nb_data` object again. But you can use the `nb_readExcelMoreSheets` function to read it to an `nb_DataCollection` object. For more information on the `saveDataBase` look it up in the [library](#).

5.3.8 Writing data to other file formats

The method `saveDataBase` makes it also possible to write an `nb_data` object to a `.mat` file or `.txt` file. This can be done with the 'ext' option:

```
% Save the data to a .mat file
obj = nb_data(ones(8,2),'filename','1',{'Var1','Var2'});
obj.saveDataBase('saveName','filename','ext','mat')

% Save the data to a .txt file
obj = nb_data(ones(8,2),'filename','1',{'Var1','Var2'});
obj.saveDataBase('ext','txt','saveName','filename')
```

From the example you can see that it is possible to save data of the object to a `.txt` ('`txt`') and `.mat` ('`mat`'), these are the only file formats supported besides excel ('`xlsx`').

5.3.9 Converting to other data types (objects)

In case you want to transform the `nb_data` object to any other MATLAB object/type. The `nb_data` class has some methods for doing this. See the examples below for how to do this.

```
data = nb_data([2,2,2;2,2,2],'', '1', {'Var1','Var2','Var3'});

% Convert to a structure
s = data.toStructure()

% Convert to a cell
c = data.asCell()

% Convert to a double
d = data.double()

% Convert to a nb_ts object
d = data.tonb_ts()

% Convert to a nb_cs object
cs = data.tonb_cs();
```

5.3.10 Mathematical operators

Mathematical operators act on `nb_data` objects in a special way. The operators `.*`, `.^`, `+`, `-` and `./` will try to match the variables found in both objects and do the calculation on these variables only. The rest of the variables will be given `nan` values for all data points. So for examples if an `nb_data` object which includes the variables `var1` and `var2` and another object includes only `var1`, then the operator will only act on the data of the variable `var1` of the returned `nb_data` object while the `var2` will have all `nan` values. When the observations of the two objects does not match all the data points outside the window of both object will also be returned as `nan` values.

The operators `*`, `^`, `+`, `-` and `/` act on all the data of the object, but they are only defined when combined with scalars. See the examples below.

```
data1 = nb_data(ones(2,3)*3,'','1',{'Var1','Var2','Var3'});
data2 = nb_data(ones(2,3)*2,'','1',{'Var1','Var2','Var3'});

% Element-wise operators
data = data1-data2      % Element-wise minus
data = data1+data2      % Element-wise plus
data = data1.*data2     % Element-wise multiplication
data = data1.^data2     % Element-wise power
data = data1./data2     % Element-wise division
```

```
% Operators that act on all the data of the object
data = data1-2           % Minus (Only for scalars)
data = data1+2           % Plus (Only for scalars)
data = data1*2           % Multiplication (Only defined when
                        % multiplied with a scalar)
data = data1^2           % Power (Only defined when taken power with
                        % a scalar)
data = data1/2           % Only defined when divided by a scalar
```

5.3.11 Logical operators

The `nb_data` also supports the logical operators `|`, `>`, `>=`, `==`, `<=`, `<` & `~=`. All these operators will try to match the variables found in the objects and do the test element-wise on the data of the matching variables. Contrary to the mathematical operators these operators will not work as long as the two objects do not include the same variables and dates. Examples are found below.

```
data1 = nb_data(logical([ones(2,2), zeros(2,1)]), '', '1', {'Var1', 'Var2', 'Var3'});
data2 = nb_data(logical([ones(2,1), zeros(2,2)]), '', '1', {'Var1', 'Var2', 'Var3'});

data = data1 & data2
data = data1 | data2

data1 = nb_data(ones(2,3)*3, '', '1', {'Var1', 'Var2', 'Var3'});
data2 = nb_data(ones(2,3)*2, '', '1', {'Var1', 'Var2', 'Var3'});

data = data1 > data2
data = data1 >= data2
data = data1 == data2
data = data1 <= data2
data = data1 < data2
data = data1 ~= data2
```

5.3.12 Other useful methods

There are also many other methods of the class which could make programming with dimensionless data more easy. I will not describe these methods here, but instead provide a list with links to the library at the end of this reference manual.

- [addVariable](#) Add a new variable to the object
- [deleteVariables](#) Delete some variables of the object
- [isempty](#) Checks if an object is empty
- [keepVariables](#) Keep some variables of the object
- [rename](#) Rename some variables or types of the object
- [setValue](#) Set the value of some data of the object
- [structure2Dataset](#) Add a data set from a structure
- [window](#) Shrink the window of the object

5.3.13 Examples

See the example script file found at \\NBTOOLBOX\\Examples\\DataManagement\\nb_dataExamples.m

5.4 Import multi-paged data

Some models can e.g. be estimated on real-time data. In the toolbox such data are stored as multi-paged datasets. The toolbox provide you two functions for making this process easier for you.

5.4.1 Excel spreadsheet

To import multi-paged dataset from an excel spreadsheet you can use the `nb_readExcelMorePages` function. Each worksheet must have the same format, for more on the needed format see section 1, 2 or 3. An example follows:

```
data1 = nb_readExcelMorePages('test.xlsx');
data2 = nb_readExcelMorePages('test.xlsx',sorted,'1',{'Sheet1','Sheet2'});
```

This function will create a link to the excel spreadsheet if the full path is provided.

5.4.2 A note on real-time data

To secure that your real-time data is structured correctly to be used by the different model objects provided to you by this toolbox, you should use the `cleanRealTime` method.

5.5 Create a link to the data source

The `nb_data`, `nb_ts` and `nb_cs` classes have an update method which will make it possible to create a link to the data source. The data source can be an excel spreadsheet, or a .mat file. To make the object updateable with the update method you need to provide the full path when you initialize the `nb_data`, `nb_ts` or `nb_cs`.

Be aware that even if you do transformations on the objects, by for example the egrowth method, the object is still updateable. I.e. it will read the source and do the method calls again on the updated data.

If the data source(s) of a `nb_graph_data`, `nb_graph_ts`, `nb_graph_cs`, `nb_graph_adv` or `nb_graph_package` object is (are) updatable, i.e. given by (an) updatable `nb_data`, `nb_ts` and `nb_cs` object(s), then that object is updatable too.

6 Graphics

With this toolbox you can create varieties of graphs. Be it line, bar, area or a combination of them. This could be done with the classes `nb_graph_data`, `nb_graph_ts` and `nb_graph_cs`, for dimensionless data, time-series data and cross sectional (case) data respectively. Given that these three classes do not share all the same functionalities they will be described separately.

6.1 Time-series data

In this section we will discover many of the possibilities of the `nb_graph_ts` class. First we need to provide the data of the graph. The data must be stored in an `nb_ts` object. A example of how this is done is provided below. In this example the data are loaded from an excel spreadsheet to an `nb_ts` object. To initialize an `nb_graph_ts` object you only need to provide the loaded `nb_ts` object as the input to the command `nb_graph_ts`.

```
%% Reading data from a excel spreadsheet
data = nb_ts('example_ts_quarterly');

% Initialize a nb_graph_ts object with the above data
plotter = nb_graph_ts(data);
```

Be aware that the `nb_graph_ts` object is a handle object. Which means that you do not need to assign changes to the object.

When this is done you can in practice graph the data, but the output graph may be unsatisfactory, because the default settings are used when plotting the data. To set the appearance of the graph you can use the `set` method of the `nb_graph_ts` class. The class has many different properties which will change the way the data are plotted. To set properties of the object use the `set` method. You must provide one or more property name, property value pair(s) of arguments. See the example below.

```
% Set one property of the object
plotter.set('title','test');

% Set more properties of the object at one function call
plotter.set('title','test','titleFontSize',14);
```

To get a list of all the properties of the `nb_graph_ts` class you can type `properties(nb_graph_ts)` in the command line. If you want some help on one property you can type `help nb_graph_ts.<propertyName>`. E.g. `help nb_graph_ts.title`. You can also look up the property in the `archive`. To get a list of the methods of the `nb_graph_ts` class type `methods(nb_graph_ts)` in the command line. If you want some help on some method you can type `help nb_graph_ts.<methodName>`. E.g. `help nb_graph_ts.set`.

We are now ready to do some plotting, but the `nb_graph_ts` has more graphing methods. These methods will now be described and some examples will be provided.

6.1.1 The method `graph()`

The method `graph` will plot the variables given in the input `nb_ts` object or the variables given by the `variablesToPlot` property in one graph. (If the `nb_ts` object has more pages, each page will be plotted in separate graphs.) The default plot type is line plot. So if we continue on our example above and just type `plotter.graph` the graph given in the figure 1 is plotted.

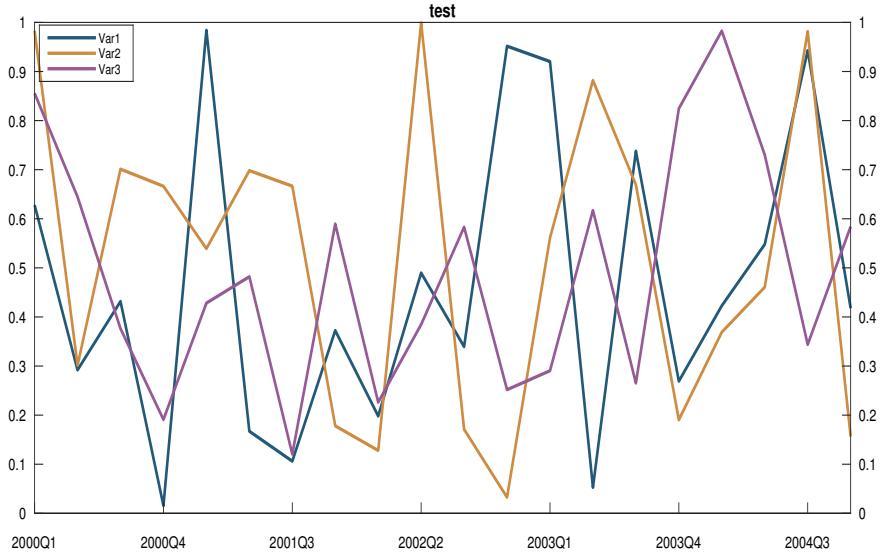


Figure 1: Simple line plot.

6.1.1.1 Different plot types of the graph method The `nb_graph_ts` class supports different plot types, which you can set with the property `plotType`. The supported types are :

- 'area' Area plot(s)
- 'candle' Candle plot(s)
- 'dec' Decomposition plot(s). Which is a bar plot with also a line with the sum of the stacked bars.
- 'grouped' Grouped bar plot(s)
- 'line' Line plot(s)
- 'scatter' Scatter plot(s)
- 'stacked' Stacked bar plot(s)

Here follows examples of the different plot types.

```
% Adding data to an object of class nb_ts
data = nb_ts([2,2;1,3;3,2],'', '2012', {'Var1', 'Var2'});

% Initialize an nb_graph_ts object with the above data
plotter = nb_graph_ts(data);

% Set the plotType to line
plotter.set('plotType', 'line');

% Plot the object
plotter.graph

% Set the plotType to stacked
plotter.set('plotType', 'stacked');

% Plot the object
plotter.graph

% Set the plotType to grouped
plotter.set('plotType', 'grouped');

% Plot the object
plotter.graph

% Set the plotType to dec
plotter.set('plotType', 'dec');

% Plot the object
plotter.graph

% Set the plotType to area
plotter.set('plotType', 'area');

% Plot the object
plotter.graph

% Set the plotType to candle (You must use the
% candleVariables property)
data = nb_ts([1,2,3,4,2.5;1,2,3,4,2.5;1,2,3,4,2.5],'', ...
    '2012', {'var1', 'var2', 'var3', 'var4', 'var5'});
plotter = nb_graph_ts(data);

plotter.set('candleVariables', {'close', 'var2', 'open', 'var3'}, ...
    'plotType', 'candle', ...
```

```

        'yLim',           [0 5]);
plotter.graph();

% Set the plotType to scatter (The default is to use the two
% first variables of the data)
data = nb_ts(rand(10,4),'', '2010', {'var1','var2','var3','var4'});
plotter = nb_graph_ts(data);

plotter.set('plotType','scatter',...
'scatterDates',{'Group1',{'2010','2019'}});
plotter.graph()

```

See the section [6.5](#) for more examples.

6.1.2 The method graphSubPlots()

The method `graphSubPlots` will plot each variable stored in the input data or all variables given by the property `variablesToPlot` in separate subplots. To set the number of subplots on each figure use the `subPlotSize` property. It must be set to a `1x2` double. The first number will be the number of rows of the subplots and the second number will be the number of columns of the subplot. This method is useful when you need a quick view of the data stored in an `nb_ts` object, or when you want to plot different pages (data sets) of the `nb_ts` object against each other quickly. See the examples:

```

% One paged data
data      = nb_ts(rand(20,4), 'Data1', '2012Q1',...
                   {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Graph all the variables stored in the data object in separate
% subplots
plotter.graphSubPlots();

% Two paged data
data      = nb_ts(rand(20,4,2), {'Data1','Data2'}, '2012Q1',...
                   {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Graph all the variables stored in the data object in separate
% subplots
plotter.graphSubPlots();

```

The plotted two paged data gives the figure [2](#).

6.1.2.1 Different plot types of the graph method The method `graphSubPlots` supports different plot types, which you can set with the property `plotType`. The supported types are:

- 'area' Area plot(s)
- 'line' Line plot(s)
- 'stacked' and 'grouped' Bar plot(s)

6.1.3 The method graphInfoStruct()

This method can be used to set up a graph package. This is especially important when large output from models are plotted. Instead of using `variablesToPlot`, and other properties of the `nb_graph_ts` class you must use the `GraphStruct` property. This property must be either set to a MATLAB struct or a .m file in a specific format. We will start with how to use the struct option. Here is an example:

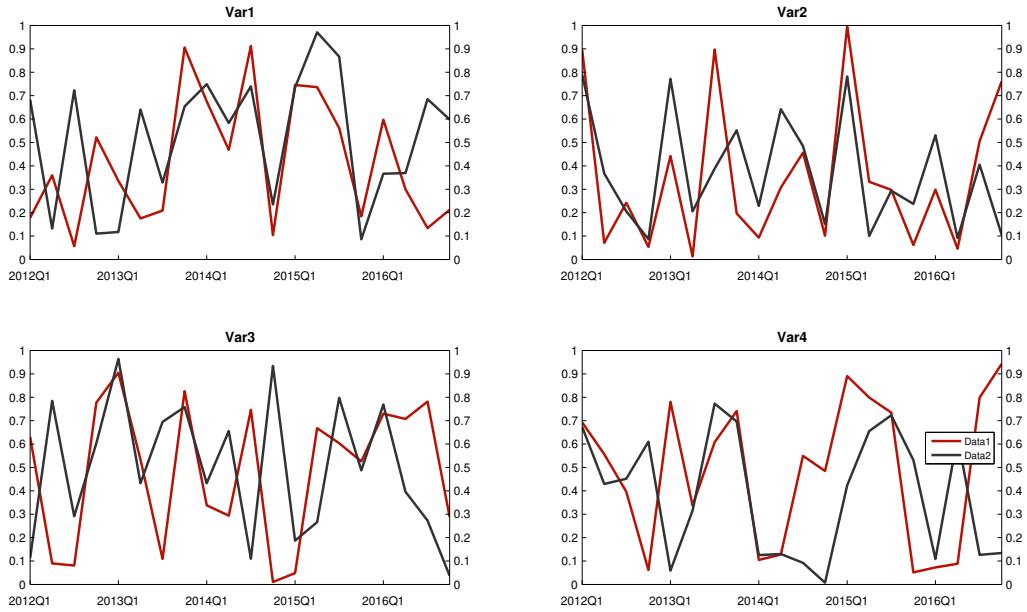


Figure 2: A figure produced by the graphSubPlots method.

```
% Two paged data
data      = nb_ts(rand(20,4,2),{'Data1','Data2'},'2012Q1',...
                  {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Set up the graph settings
s = struct();
s.Example = {
    'Var1',      {'ylim', [-1 1],'ySpacing',1};
    'Var3*100',  {'yLabel','Prosent'};
    'Var3./Var4', {'yLabel','Prosent','title','Expression (Var4./Var5)'};
    'Var4',       {}};

% Set the graphStruct property
plotter.set('graphStruct',s);

% Graph given the information given by the GraphStruct property
plotter.graphInfoStruct();
```

We must first initialize the `nb_graph_ts` object in the normal way. Then we construct a variable `s` representing a struct. Then we add a field called `Example`, which will represent one figure. This field must be a cell with size $Z \times 2$. The number of rows (Z) will decide the number of subplots of the produced figure. I.e. each row will result in one subplot. The first element of the row must be the name of the variable or the a mathematical expression to be plotted. Here the expression can include all operators and methods defined by the `nb_math_ts` class.¹⁶ The second element must again be a cell array. Here you can provide the optional inputs of the given subplot (as input name, input value combinations). The following inputs are supported:

¹⁶Inside the method `graphInfoStruct` the `createVariable` method of the `nb_ts` class will be called for each expressions it does not find in the variable names.

- 'colorOrder' Sets the color order of the specific subplot
- 'legends' Sets the legend text of the specific subplot
- 'legLocation' Sets the location of the legend of the specific subplot
- 'legPosition' Sets the position of the legend of the specific subplot
- 'lineStyle' Sets the line style of the specific subplot
- 'plotType' Sets the plot type of the specific subplot
- 'title' Sets the title of the specific subplot
- 'xLabel' Sets the x-axis label of the specific subplot
- 'yLabel' Sets the y-axis label (left) of the specific subplot
- 'yLabelRight' Sets the y-axis label (right) of the specific subplot
- 'yLim' Sets the y-axis (left or both) limits of the specific subplot
- 'yLimRight' Sets the y-axis (right) limits of the specific subplot
- 'ySpacing' Sets the y-axis (left or both) spacing of the specific subplot
- 'ySpacingRight' Sets the y-axis (right) spacing of the specific subplot

If you want the graphInfoStruct method to produce more figures you can add more fields in the same way as described above, but you must give the fields different names. When you are done with constructing the struct (s in our example) you must assign it to the property GraphStruct. This is done with the set method as shown in the example above. When that is done you can create the graph(s) by calling the `graphInfoStruct` method. As already mentioned, it is also possible to set the property GraphStruct to the name of a .m file, as a string without the extension. In the file you must take it for given that the obj.GraphStruct is a struct, so you can add different fields to this "variable" in the .m file you provide. An example of a .m file on this format is provided below:

```
%-----
% Each field gives one figure
%-----
obj.GraphStruct.ExampleI = {
    'Var1',      {'ylim', [-1 1], 'ySpacing', 1};
    'Var3*100',  {'yLabel', 'Prosent'};
    'Var3./Var4', {'yLabel', 'Prosent', 'title', 'Expression (Var3./Var4)'};
    'Var4',       {}};
%-----
obj.GraphStruct.ExampleII = {
    'Var2',      {'ylim', [-1 1], 'ySpacing', 1};
    'Var3*100',  {'yLabel', 'Prosent'};
    'Var2./Var4', {'yLabel', 'Prosent', 'title', 'Expression (Var2./Var4)'};
    'Var4',       {}};
%-----
```

And this code will use the .m file:

```
% Two paged data
data      = nb_ts(rand(20,4,2),{'Data1','Data2'},'2012Q1',...
                  {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Set the graphStruct property
plotter.set('graphStruct','graphInfoFile');
```

```
% Graph all the variables stored in the data object in separate
% subplots
plotter.graphInfoStruct();
```

Be aware that many of the properties of the `nb_graph_ts` class are supported by this method. See the archive for more on these properties.

We will now look at some properties which will set the more general settings of the graphs.

6.1.4 Set the spacing between the x-axis tick marks

To set the spacing between the x-axis tick mark labels, you can use the `spacing` property of the `nb_graph_ts` class. The property must be set to an integer. The integer will be the number of periods given in the frequency of the data or the frequency set by the property `xTickFrequency` (See the next paragraph). Here is an example:

```
data      = nb_ts(rand(40,2),'', '2012Q1', {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Uses the property spacing
plotter.set('spacing',4);

% Graph
plotter.graph
```

6.1.5 Set the frequency of the x-axis tick marks

Say you want to plot quarterly data, but want the frequency of the x-axis tick mark labels to be yearly. You can do this using the `xTickFrequency` property of the `nb_graph_ts` class. E.g.

```
data      = nb_ts(rand(40,2),'', '2012Q1', {'Var1','Var2'});
% Initialize an nb_graph_ts object with the above data
plotter = nb_graph_ts(data);

% Uses the property xTickFrequency to do this
plotter.set('xTickFrequency','yearly');

% Graph
plotter.graph
```

Which will produce figure 3. It is only possible to convert the x-axis frequency to a lower one. The supported frequencies are:

- 'daily'
- 'monthly'
- 'quarterly'
- 'semiannually'
- 'yearly'

6.1.6 Set the start date of the x-axis tick marks

To set the start date of the x-axis tick mark labels you can use the `xTickStart` property of the `nb_graph_ts` class. It must either be set to a string with the date or an object which is of an subclass of the `nb_date` class. Here is an example:

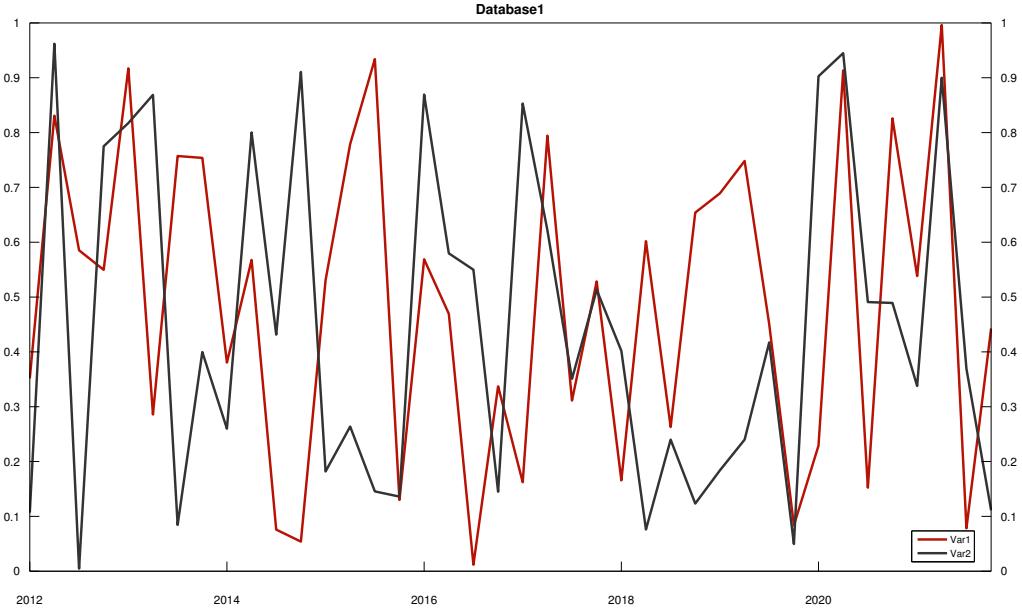


Figure 3: Use of the xTickFrequency property.

```

data      = nb_ts(rand(40,2),'', '2012Q1', {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Uses the property xTickStart to do this
plotter.set('xTickStart','2012Q2', 'spacing', 4);

% Graph
plotter.graph

```

6.1.7 Set how to interpreter the data

You can also set how to interpret the data by setting the property dateInterpreter. I.e. where to place the data points in relation to the tick mark labels. You have the option:

- 'end' The data are given at the end of the period
- 'middle' The data are given at the middle of the period
- 'start' The data are given at the start of the period

6.1.8 Set the alignment of the x-axis tick mark labels

You can set the alignment of the x-axis tick mark labels to move them to be in between the x-axis tick marks. Just set the property xTickLabelAlignment to 'middle' in combination with the dateInterpreter set to 'middle'. E.g:

```

data      = nb_ts([2,2;-1,3;3,-2;3,4;-2,3;1,2],'', '2012Q1',...
                  {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Uses the property xTickLabelAlignment to do this.
plotter.set('dateInterpreter',      'middle',...
            'plotType',           'grouped',...

```

```

    'xTickLabelAlignment', 'middle');

% Graph
plotter.graph

```

This code will result in figure 4.

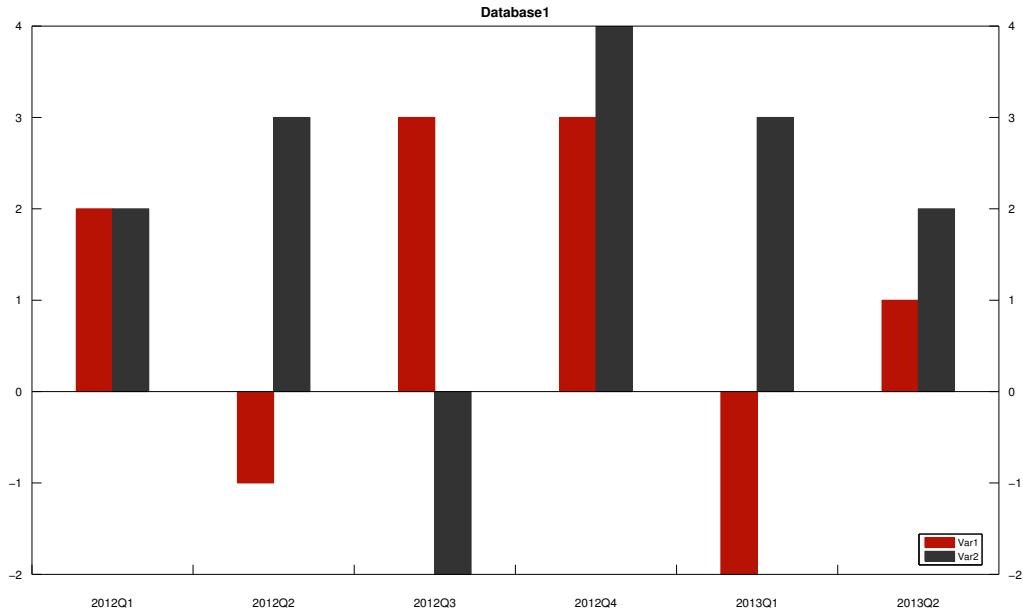


Figure 4: The xTickLabelAlignment property set to 'middle'.

6.1.9 Set the location of the x-axis tick marks

This could be done with setting the property xTickLocation. It can be set to 'bottom', 'top' or a scalar value for where to place the x-axis tick marks. Below is an example where we place the x-axis tick marks on the baseline (which is at zero) of a bar plot:

```

data      = nb_ts([2,2;-1,3;3,-2;3,4;-2,3;1,2], '', '2012Q1',...
                  {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Uses the property xTickLocation to do this
plotter.set('dateInterpreter',      'middle',...
            'plotType',           'grouped',...
            'xTickLabelAlignment', 'middle',...
            'xTickLocation',       0);

% Graph
plotter.graph

```

6.1.10 How to interpret missing data

If you plot data with missing values, the graph can end up looking unsatisfactory. With the missingValues property of the `nb_graph_ts` class you can fix this. I.e. you can set how to interpret the missing values. You have three options:

- 'interpolate' Linearly interpolate the missing values
- 'strip' Strip the missing values
- 'both' First strip up until a date given by the property stopStrip and then interpolate the rest of the missing values. If the stopStrip property is not given this will result in the same as the 'strip' option

Here are some examples:

```
% Create data with missing observations
data = nb_ts.rand('2019M3D1',100,1);
data(2:7:end,:) = nan;
data(3:7:end,:) = nan;
plotter = nb_graph_ts(data);
plotter.graph()

% Interpolate the missing observations
plotter.set('missingValues','interpolate');
plotter.graph()

% Strip the missing observations
plotter.set('missingValues','strip');
plotter.graph()

% Generate data with missing observations (Only workdays)
dataDaily = nb_ts.rand('2016M3D1',1000,1);
dataDaily(2:7:end,:) = nan;
dataDaily(3:7:end,:) = nan;

% Add some artificial forecast
dataQuarterly = nb_ts(rand(3,1),'', '2019Q1', {'Var1'});
data = dataDaily.merge(dataQuarterly);
plotter = nb_graph_ts(data);

% Interpolate the missing observations works fine
plotter.set('missingValues','interpolate');
plotter.graph()

% Strip the missing observations does not
plotter.set('missingValues','strip');
plotter.graph()

% Solution use the 'both' option in combined with the stopStrip
% command
plotter.set('missingValues','interpolate','stopStrip','2013M9D30');
plotter.graph()
```

6.1.11 Creating a fan chart

To create a fan chart you can utilize the fanDatasets property of the `nb_graph_ts` class. To set the percentiles to calculate you can set the fanPercentiles property. Here is an example:

```
% Creating some artificial data
data = nb_ts([ones(20,1,100);ones(16,1,100)*0.5 + ...
rand(16,1,100)],'', '2008Q1', 'Var1');

% Get the mean (which must be given to the nb_graph_ts constructor)
meanData = mean(data,'nb_ts',3);
```

```

meanData = meanData.window('','',{},{},1);
plotter = nb_graph_ts(meanData);

% Set the fanDatasets property
plotter.set('fanDatasets',data);

% Graph the fan chart
plotter.graph();

```

It is also possible to add a fan legend. You can do this by setting the fanLegend property to 1. If you want to change the location of the fan legend you can use the fanLocation property, and if you want to change the text of the fan legend you can use the fanLegendText¹⁷. Here is an example:

```

% Creating some artificial data
data      = nb_ts([ones(20,1,100);ones(16,1,100)*0.5 + ...
    rand(16,1,100)],'', '2008Q1', 'Var1');

% Get the mean (which must be given to the nb_graph_ts constructor)
meanData = mean(data,'nb_ts',3);
meanData = meanData.window('','',{},{},1);
plotter = nb_graph_ts(meanData);

% Set the fanDatasets property
plotter.set('fanDatasets', data, ...
    'fanLegend', 1, ...
    'fanLegendLocation', [0.3,0.7]);

% Graph the fan chart
plotter.graph();

```

6.1.12 Adding highlighted areas

With the use of the highlight property of the `nb_graph_ts` class you can add a colored area behind the plotted variables. It must be set to a cell on the form `{'startDate','endDate'}` if just one highlighted area is wanted or a nested cell `{{'startDate1','endDate1'},...}` if more highlighted areas are wanted. To set the color of the highlighted area(s) you can use the highlightColor property. It must be a cell array with the colors of the highlighted area(s). Each element must either be a string with the color name or a 1x3 double with the RGB color specification. The cell array must match the size of the highlight property, i.e. the number of highlighted areas plotted. The highlightColor property is only optional. The default color of the highlighted areas is light blue. E.g:

```

data      = nb_ts(rand(36,2)*3,'','2008Q1',{'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set some properties of the graph to create one highlighted area
plotter.set('highlight', {'2012Q1','2014Q1'}, ...
    'highlightColor', {'orange'});

plotter.graph()

% Set some properties of the graph to create more highlighted areas
plotter.set('highlight', {'2011Q1','2012Q1'}, ...
    {'2013Q1','2014Q1'}, ...
    'highlightColor', {'orange','light blue'});

plotter.graph()

```

Figure 5 is produced by the second example given above.

¹⁷Default are the percentiles given by the fanPercentiles property.

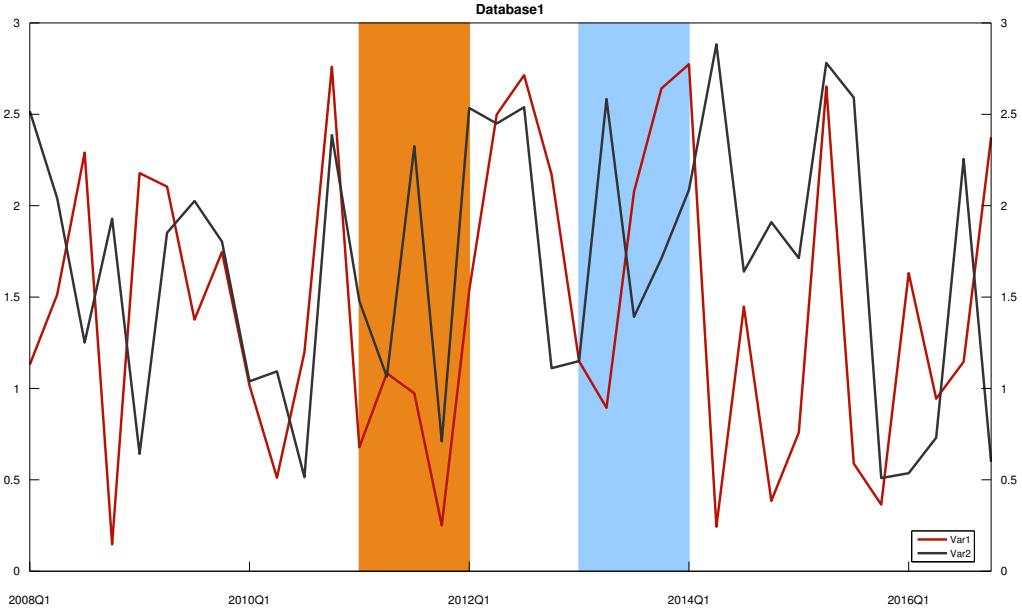


Figure 5: Adding highlighted areas to the figure.

6.2 Cross-sectional data

To plot cross-sectional data you can utilize the `nb_graph_cs` class. To initialize an `nb_graph_cs` class, you first need to provide an object of class `nb_cs` as input to the `nb_graph_cs` command. I.e:

```
% Reading data from a excel spreadsheet to construct a nb_cs object
data = nb_cs('example_cs');

% Initialize a nb_graph_cs object
plotter = nb_graph_cs(data)
```

Be aware that the `nb_graph_cs` object is a handle object. Which means that you do not need to assign changes to the object.

To set the properties of the `nb_graph_cs` object you can use the `set` method of the `nb_graph_cs` class. E.g:

```
% Set one property of an nb_graph_cs object
plotter.set('title','A title');

% Set more properties of an object of class nb_graph_cs
plotter.set('title','A title','titleFontSize',14);
```

In the first example we are assigning the value 'A title' to the property `title` of the `nb_graph_cs` object. As the second example shows, it is also possible to set more properties of an object at the same time. I.e. provide more property name, property value combinations.

To get a list off all the properties of the `nb_graph_cs` class you can type `properties(nb_graph_cs)` in the command line. If you then want help on one property you can type `help nb_graph_cs.<propertyName>`. E.g. `help nb_graph_cs.title`. You can also see the `archive`. To get a list of the methods of the `nb_graph_cs` class type `methods(nb_graph_cs)` in the command line. If you want help on some method you can type `help nb_graph_cs.<methodName>`. E.g. `help nb_graph_cs.set`.

6.2.1 The method graph()

The `graph` method can be used to create graphs. This method can be compared with `graph` method of the `nb_graph_ts` class. We will start with a simple example:

```
% Reading data from a excel spreadsheet to construct a nb_cs object
data = nb_cs('example_cs');

% Initialize a nb_graph_cs object
plotter = nb_graph_cs(data);

% Do the graphing
plotter.graph()
```

Which will result in figure 6.

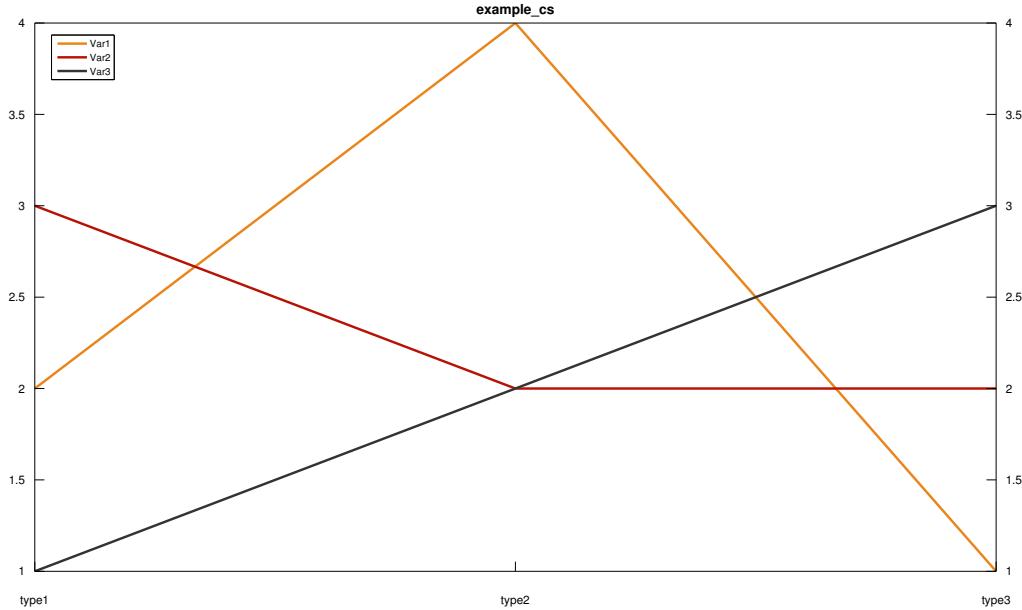


Figure 6: A simple graph using the `nb_graph_cs` class.

6.2.1.1 Different plot types of the graph method The `nb_graph_cs` class supports different plot types, which you can set with the property `plotType`. The supported types are :

- 'area' Area plot(s)
- 'candle' Candle plot(s)
- 'dec' Decomposition plot(s). Which is a bar plot with also a line with the sum of the stacked bars.
- 'grouped' Grouped bar plot(s)
- 'line' Line plot(s)
- 'pie' Pie plot(s)
- 'radar' Radar plot(s)
- 'scatter' Scatter plot(s)
- 'stacked' Stacked bar plot(s)

Here follows examples of the different plot types.

```

%% Line plot
data      = nb_cs([2,2;1,3;3,2],'',{'type1','type2','type3'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.graph();

%% Area plot
data      = nb_cs([2,2;1,3;3,2],'',{'type1','type2','type3'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('plotType', 'area');
plotter.graph();

%% Stacked bar plot
data      = nb_cs([2,2;1,3;3,2],'',{'type1','type2','type3'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('plotType', 'stacked');
plotter.graph();

%% Grouped bar plot
data      = nb_cs([2,2;1,3;3,2],'',{'type1','type2','type3'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('plotType', 'grouped');
plotter.graph();

%% Horizontal stacked bar plot
data      = nb_cs([2,2;1,3;3,2],'',{'type1','type2','type3'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('plotType', 'stacked', 'barOrientation', 'horizontal');
plotter.graph();

%% Pie plot
data      = nb_cs([2,2,3,2],'',{'type1'},...
                  {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_cs(data);
plotter.set('plotType', 'pie');
plotter.graph();

%% Radar plot
data      = nb_cs([1,2; 2,4; 3,5; 2,1; 3,4; 2,5],'',...
                  {'type1','type2','type3','type4','type5','type6'},...

```

```

        {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('plotType','radar');
plotter.graph();

%% Candle plot
data = nb_cs([1,2,3,4,2.5;1,2,3,4,2.5;1,2,3,4,2.5],'',...
    {'type1','type2','type3'},{'var1','var2','var3','var4','var5'});
plotter = nb_graph_cs(data);

plotter.set('candleVariables',{'low','var1','close','var2',...
    'open','var3','high','var4','indicator','var5'},...
    'plotType',           'candle');
plotter.graph();

%% Scatter plot
data = nb_cs(rand(2,10),'',{'type1','type2'},{'var1','var2',...
    'var3','var4','var5','var6','var7','var8','var9','var10'});

plotter = nb_graph_cs(data);
plotter.set('plotType','scatter',...
    'scatterVariables',{'scatterGroup1',{'var1','var2','var3',...
    'var4','var5','var6','var7','var8','var9','var10'}});
plotter.graph()

```

See the section [6.6](#) for more examples.

6.2.2 The method graphSubPlots()

The method `graphSubPlots` will plot each variable stored in the input data or all variables given by the property `variablesToPlot` in separate subplots. To set the number of subplots on each figure use the `subPlotSize` property. It must be set to a `1x2` double. The first number will be the number of rows of the subplots and the second number will be the number of columns of the subplot. This method is useful when you need a quick view of the data stored in an `nb_cs` object, or when you want to plot different pages (data sets) of the `nb_cs` object against each other quickly. See the examples below:

```

% Initialize multi-paged nb_cs object
data = nb_cs(rand(3,3,3),'',{'type1','type2','type3'},...
    {'var1','var2','var3'});

plotter = nb_graph_cs(data);

% Graph all variables in separate subplots
plotter.graphSubPlots();

```

6.2.2.1 Different plot types of the graph method The method `graphSubPlots` supports different plot types, which you can set with the property `plotType`. The supported types are:

- 'area' Area plot(s)
- 'line' Line plot(s)
- 'stacked' and 'grouped' Bar plot(s)

6.2.3 The method graphInfoStruct()

This method can be used to set up a graph package. This is especially important when large output from models are plotted. Instead of using `variablesToPlot`, and other properties of the `nb_graph_cs` class you must use the `GraphStruct` property. This property

must be either set to a MATLAB struct or a .m file in a specific format. We will start with how to use the struct option. Here is an example:

```
% Three paged data
data = nb_cs(rand(3,3,3),'',{'type1','type2','type3'},...
              {'var1','var2','var3'}); 

plotter = nb_graph_cs(data);

% Set up the graph settings
s = struct();
s.Example = {
    'var1',      {'ylim', [-1 1]}, 'ySpacing', 1;
    'var3*100',  {'yLabel', 'Present'};
    'var3./var2', {'yLabel', 'Present', 'title', 'Expression (Var3./var2)'};
    'var2',      {}};

% Set the graphStruct property
plotter.set('graphStruct',s);

% Graph all the variables stored in the data object in separate
% subplots
plotter.graphInfoStruct();
```

We must first initialize the `nb_graph_cs` object in the normal way. Then we construct a variable `s` representing a struct. Then we add a field called `Example`, which will represent one figure. This field must be a cell with size $Z \times 2$. The number of rows (Z) will decide the number of subplots of the produced figure. I.e. each row will result in one subplot. The first element of the row must be the name of the variable or the a mathematical expression to be plotted. Here the expression can include all operators and methods defined by the MATLAB double class.¹⁸ The second element must again be a cell array. Here you can provide the optional inputs of the given subplot (as input name, input value combinations). The following inputs are supported:

- 'colorOrder' Sets the color order of the specific subplot
- 'legends' Sets the legend text of the specific subplot
- 'legLocation' Sets the location of the legend of the specific subplot
- 'legPosition' Sets the position of the legend of the specific subplot
- 'lineStyle' Sets the line style of the specific subplot
- 'plotType' Sets the plot type of the specific subplot
- 'title' Sets the title of the specific subplot
- 'xLabel' Sets the x-axis label of the specific subplot
- 'yLabel' Sets the y-axis label (left) of the specific subplot
- 'yLabelRight' Sets the y-axis label (right) of the specific subplot
- 'yLim' Sets the y-axis (left or both) limits of the specific subplot
- 'yLimRight' Sets the y-axis (right) limits of the specific subplot
- 'ySpacing' Sets the y-axis (left or both) spacing of the specific subplot
- 'ySpacingRight' Sets the y-axis (right) spacing of the specific subplot

¹⁸Inside the method `graphInfoStruct` the `createVariable` method of the `nb_cs` class will be called for each expressions it does not find in the variable names.

If you want the graphInfoStruct method to produce more figures you can add more fields in the same way as described above, but of course you must give the fields different names. When you are done with constructing the struct (s in our example) you must assign it to the property GraphStruct. This is done with the set method as shown in the example above. When that is done you can create the graph(s) by calling the `graphInfoStruct` method. As already mentioned it is also possible to set the property GraphStruct to a name of a .m file, as a string without the extension. See the section [6.1.3](#) for more on how to set up this file.

Be aware that many of the normal properties of the `nb_graph_cs` class are also supported by this method. See the library for more on this properties.

6.2.4 Multi line x-axis tick mark labels

Sometimes you will end up in a situation where the type names are too long to keep on one line. I.e. they will overlap in the plotted figure. To fix this problem you can use the `xTickLabels` or the `lookUpMatrix` property of the `nb_graph_cs` class.

6.2.4.1 Using the `xTickLabels` property The `xTickLabels` property must be set to a cell array of strings or chars. I.e. each element must either be a string or a multi-row char. Each element with a multi-row char will result in a multi-lined x-axis tick mark label with the same number of lines as the number of rows of the provided char. See the examples below:

```
% Using the xTickLabels property
data      = nb_cs([1,2; 2,4; 3,5; 2,1; 3,4; 2,5; 4,5; 3,4],'',...
                  {'type1','type2','A too long type name',...
                   'Another too long type name','type5','type6',...
                   'type7','type8'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);

% Now the x-axis tick mark labels will overlap
plotter.set('axesFontSize',16);
plotter.graph

plotter.set('xTickLabels',{'type1','type2',...
                           char('A too long','type name'),...
                           char('Another too long','type name'),...
                           'type5','type6','type7','type8'});
plotter.graph
```

6.2.4.2 Using the `lookUpMatrix` property The `lookUpMatrix` property must either be set to a MATLAB cell or a .m file on a specific format. In the example that follows a cell will be used:

```
data      = nb_cs([1,2; 2,4; 3,5; 2,1; 3,4; 2,5; 4,5; 3,4],'',...
                  {'type1','type2','A too long type name',...
                   'Another too long type name','type5','type6',...
                   'type7','type8'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);

% Now the x-axis tick mark labels will overlap
plotter.set('axesFontSize',16);
plotter.graph

% Set the lookUpMatrix property to make the x-axis tick mark labels
% multi-lined
s={'A too long type name',    char('A too long','type name'),''
```

```
'Another too long type name',char('Another too long','type name'),''};  
  
plotter.set('lookUpMatrix',s);  
plotter.graph
```

If you instead want to use a .m file, the file must contain something like:

```
obj.lookUpMatrix = {  
    'A too long type name',    char('A too long','type name'), ''  
    'Another too long type name',char('Another too long','type name'), ''  
};
```

And to use this .m file use the following code (Be aware that the extension should not be provided):

```
data      = nb_cs([1,2; 2,4; 3,5; 2,1; 3,4; 2,5; 4,5; 3,4],'',...  
                  {'type1','type2','A too long type name',...  
                   'Another too long type name','type5','type6',...  
                   'type7','type8'},...  
                  {'Var1','Var2'});  
plotter = nb_graph_cs(data);  
  
% Here the x-axis tick mark labels will overlap  
plotter.set('axesFontSize',16);  
  
% Set the lookUpMatrix property to make the x-axis tick mark labels  
% multi lined  
plotter.set('lookUpMatrix','lookUpMatrixExample');  
plotter.graph
```

Both ways will result in the figure 7.

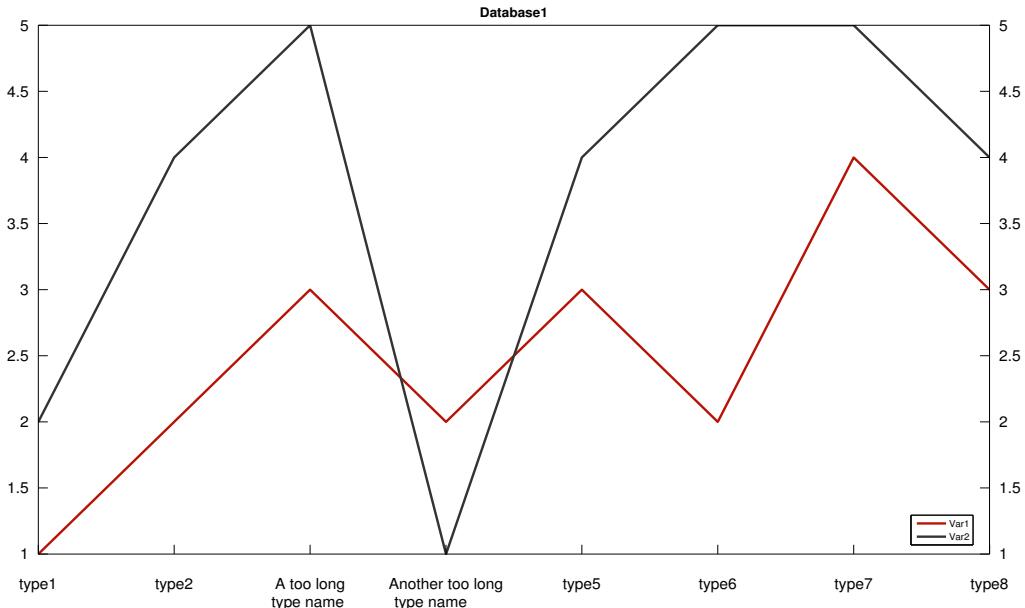


Figure 7: A figure with multi lined x-axis tick mark labels.

This way of creating multi-lined x-axis tick marks must be used if you are producing both an English and a Norwegian version at the same time. I.e. use the last column of the above cell (s or obj.lookUpMatrix) to translate the x-axis tick mark labels to Norwegian.

6.3 Dimensionless data

To plot dimensionless data you can utilize the `nb_graph_data` class. To initialize a `nb_graph_data` class, you first need to provide an object of class `nb_data` as input to the `nb_graph_data` command. I.e:

```
% Reading data from a excel spreadsheet to construct a nb_data object
data = nb_data('example_data');

% Initialize a nb_graph_data object
plotter = nb_graph_cs(data)
```

Be aware that the `nb_graph_data` object is a handle object. Which means that you do not need to assign changes to the object.

To set the properties of the `nb_graph_data` object you can use the `set` method of the `nb_graph_data` class. E.g:

```
% Set one property of an nb_graph_data object
plotter.set('title','A title');

% Set more properties of an object of class nb_graph_data
plotter.set('title','A title','titleFontSize',14);
```

In the first example we are assigning the value 'A title' to the property title of the `nb_graph_data` object. As the second example shows, it is also possible to set more properties of an object at the same time. I.e. provide more property name, property value combinations.

To get a list off all the properties of the `nb_graph_cdata` class you can type `properties(nb_graph_data)` in the command line. If you then want help on one property you can type `help nb_graph_data.<propertyName>`. E.g. `help nb_graph_data.title`. You can also see the [library](#). To get a list of the methods of the `nb_graph_data` class type `methods(nb_graph_cs)` in the command line. If you want help on some method you can type `help nb_graph_data.<methodName>`. E.g. `help nb_graph_data.set`.

6.3.1 The method `graph()`

The `graph` method can be used to create graphs. This method can be compared with `graph` method of the `nb_graph_ts` class. We will start with a simple example:

```
% Reading data from a excel spreadsheet to construct a nb_cs object
data = nb_data('example_data');

% Initialize a nb_graph_cs object
plotter = nb_graph_data(data)

% Do the graphing
plotter.graph()
```

6.3.1.1 Different plot types of the `graph` method The `nb_graph_data` class supports different plot types, which you can set with the property `plotType`. The supported types are :

- 'area' Area plot(s)
- 'candle' Candle plot(s)
- 'dec' Decomposition plot(s). Which is a bar plot with also a line with the sum of the stacked bars.
- 'grouped' Grouped bar plot(s)
- 'line' Line plot(s)
- 'scatter' Scatter plot(s)
- 'stacked' Stacked bar plot(s)

Here follows examples of the different plot types.

```

%% Line plot
data = nb_data([2,2;1,3;3,2],'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.graph();

%% Area plot
data = nb_data([2,2;1,3;3,2],'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.set('plotType', 'area');
plotter.graph();

%% Stacked bar plot
data = nb_data([2,2;1,3;3,2],'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.set('plotType', 'stacked');
plotter.graph();

%% Grouped bar plot
data = nb_data([2,2;1,3;3,2],'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.set('plotType', 'grouped');
plotter.graph();

%% Horizontal stacked bar plot
data = nb_data([2,2;1,3;3,2],'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.set('plotType', 'stacked', 'barOrientation', 'horizontal');
plotter.graph();

%% Candle plot
data = nb_data([1,2,3,4,2.5;1,2,3,4,2.5;1,2,3,4,2.5],'', ...
    '1', {'var1', 'var2', 'var3', 'var4', 'var5'});
plotter = nb_graph_data(data);

plotter.set('candleVariables', {'low', 'var1', 'close', 'var2', ...
    'open', 'var3', 'high', 'var4', 'indicator', 'var5'}, ...
    'plotType', 'candle');
plotter.graph();

%% Scatter plot
data = nb_data(rand(10,2),'', '1', {'Var1', 'Var2'});
plotter = nb_graph_data(data);
plotter.set('plotType', 'scatter', ...
    'scatterObs', {'Group1', {1,10}});
plotter.graph()

```

See the section [6.7](#) for more examples.

6.3.2 The method graphSubPlots()

The method `graphSubPlots` will plot each variable stored in the input data or all variables given by the property `variablesToPlot` in separate subplots. To set the number of subplots on each figure use the `subPlotSize` property. It must be set to a `1x2 double`. The first number will be the number of rows of the subplots and the second number will be the number of columns of the subplot. This method is useful when you need a quick view of the data stored in an `nb_data` object, or when you want to plot different pages (data sets) of the `nb_data` object against each other quickly. See the examples below:

```
% Initialize multi-paged nb_data object
data      = nb_data(rand(10,2,3), '', '1', {'var1', 'var2'});
plotter = nb_graph_data(data);

% Graph all variables in separate subplots
plotter.graphSubPlots();
```

6.3.2.1 Different plot types of the graph method The method `graphSubPlots` supports different plot types, which you can set with the property `plotType`. The supported types are:

- 'area' Area plot(s)
- 'line' Line plot(s)
- 'stacked' and 'grouped' Bar plot(s)

6.3.3 The method graphInfoStruct()

See the same section for the `nb_graph_ts` class.

6.4 Shared functionality

Now we will return the attention to properties which set some more general settings of the produced graphs.

6.4.1 Plot against different axes

To do this you can use the properties `variablesToPlot` and the `variableToPlotRight` of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class. These properties set the variables to plot against the left and the right axes respectively. Here is an example¹⁹:

```
data      = nb_ts([rand(10,1)*2, rand(10,2), rand(10,1)*2], '', '2012',...
                  {'Var1', 'Var2', 'Var3', 'Var4'});
plotter = nb_graph_ts(data);

% Set which variables to plot against which axes
plotter.set('variablesToPlot',      {'Var1', 'Var4'}, ...
            'variablesToPlotRight', {'Var2', 'Var3'})

plotter.graph()
```

¹⁹This is not supported for radar and pie plot(s). In these cases the property `variableToPlotRight` does nothing.

6.4.2 Set properties of the y-axes

To set the properties of the y-axes you can utilize the yLim, yLimRight, ySpacing, ySpacingRight, yDir, yDirRight, yScale and yScaleRight. All the xxxRight properties sets the properties of the right y-axis. The others sets the properties of both axes if the variablesToPlotRight is empty, otherwise only the left y-axis. The yLim and yLimRight properties sets the limits of the y-axes. Both must be assign a 1x2 double. E.g. [lower, upper]. The ySpacing and ySpacingRight sets the spacing between the y-axis tick marks. Both must be set to a scalar. The yDir and yDirRight properties sets the direction of the graph in the vertical dimension. Either 'normal' or 'reverse'. The properties yScale and yScaleRight sets the scale of the y-axes. Either 'normal' or 'log'. Here are some examples:

```

data      = nb_ts([rand(10,1)*2, rand(10,2), rand(10,1)*2],'', '2012',...
                  {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Plot some variables and set the y-axis limits (Here you will
% set the limits of both the left and right y-axis.)
plotter.set('variablesToPlot',      {'Var1','Var4'},...
            'yLim',                [0,2],...
            'ySpacing',             0.5);

plotter.graph()

% Set which variables to plot against which axes and set the
% y-axis limits
plotter.set('variablesToPlot',      {'Var1','Var4'},...
            'yLim',                [0,2],...
            'ySpacing',             0.5,...,
            'variablesToPlotRight', {'Var2','Var3'},...
            'yLimRight',            [0,1],...
            'ySpacingRight',         0.25);

plotter.graph()

```

The yLim and the yLimRight properties can also be given as for example [0,nan], [nan,0] or [nan,nan], where the different examples will result in setting only the lower limit, setting only the upper limit and not set any of the limits respectively.

6.4.3 Colors

To set the colors of the plot you can use the colors, colorOrder or colorOrderRight properties of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class. These must match the number of plotted variables on the left and right axes respectively, if not the default colors will be used. Be aware that you do not specify the color of the horizontal lines, vertical lines, patches or highlighted areas with these properties. The colorOrder and the colorOrderRight must be set to either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the left or right axes respectively. While the colors property must be a cellstr assigning each variable a given color. See the example below. The colors property will overload the other color properties. If not all plotted variables are assign by this property, when provided, an error will be given. Both variables plotted on the left and right axes must be given in the colors property. (The colors property is preferred.)

```
% Example of the colors property input
{'Var1',[0 1 0],'Var2','black'}
```

The supported color names are:

- 'black','svart','k' [51 51 51]/255
- 'blue','blå','b' [52 114 166]/255
- 'burgundy','burgunder' [128 0 64]/255
- 'cool grey','kald grå','cgr' [164 172 177]/255
- 'cyan','c' [0 255 255]/255
- 'deep blue','mørk blå','db' [0 60 130]/255
- 'green','grønn','g' [120 165 150]/255
- 'grey','grå','gr' [130 130 130]/255
- 'light blue','lys blå','lb' [153 204 255]/255
- 'magenta','m' [255 0 255]/255
- 'orange','o' [205 140 65]/255
- 'pink','rosa','pi' [150 115 150]/255
- 'purple','lilla','p' [100 50 100]/255
- 'red','rød','r' [221 34 65]/255
- 'sand','s' [195 185 150]/255
- 'sky blue','himmelblå','hb' [44 115 153]/255
- 'turquoise','turkis','t' [0 125 130]/255
- 'water blue','vannblå','vb' [0 60 130]/255
- 'white','hvit','w' [255 255 255]/255
- 'yellow','gul','y' [245 239 5]/255

These color names will be supported as color input to all properties of the graphing classes which have something to do with colors. The same will be the case for the RGB color specification, when given as a 1x3 double. Here are some examples:

```

data      = nb_ts([rand(10,1)*2, rand(10,2), rand(10,1)*2], '', '2012',...
                  {'Var1','Var2','Var3','Var4'});
plotter = nb_graph_ts(data);

% Using color names
plotter.set('variablesToPlot',      {'Var1','Var4'},...
            'colorOrder',        {'purple','green'}); 

plotter.graph()

% Using RGB colors
plotter.set('variablesToPlot',      {'Var1','Var4'},...
            'colorOrder',        [100, 234, 78; 34, 56, 76]/255);

plotter.graph()

% When plotting variables against two axes
plotter.set('variablesToPlot',      {'Var1','Var4'},...
            'colorOrder',        {'purple','green'},...
            'variablesToPlotRight', {'Var2','Var3'},...

```

```

    'colorOrderRight',      {'red','light blue'});
plotter.graph()

% Using the colors property instead
plotter.set('variablesToPlot',     {'Var1','Var4'},...
            'variablesToPlotRight', {'Var2','Var3'},...
            'colors',             {'Var1','purple',...
                                    'Var4','green',...
                                    'Var2','red',...
                                    'Var3','light blue'});
plotter.graph()

```

6.4.4 Edit the legend

The supported properties of the legend are:

- legAuto
- legBox
- legColor
- legColumns
- legColumnWidth
- legends
- legFontColor
- legFontSize
- legInterpreter
- legPosition
- legReorder
- legSpace

To get more information on each of the property type help nb_graph_ts.<propertyName> or nb_graph_cs.<propertyName>, where you must give the name of the property you are interested in after the dot sign.

6.4.5 Adding a fake legend

Sometimes it is needed to add extra legend information. This can be done with the fakeLegend property of the nb_graph_ts, the nb_graph_data or the nb_graph_cs classes. Here is an example:

```

data      = nb_ts(rand(36,2),'', '2008Q1', {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set some properties of the graph
plotter.set('lineStyles',{'Var1',{'-','2012Q1','--'},...
                        'Var2',{'-','2012Q1','--'}},...
            'colorOrder',{'green','orange'},...
            'legType', 'nb',...
            'legends', {'Var 1','Var 2','','','Anslag'},...
            'fakeLegend',{'Anslag',...
                         {'cData','black','lineStyle','--'}},...

```

```

'yLim', [0 1]);

plotter.graph()

```

Which will give the figure 8. To get more help on this property type `help nb_graph_ts.fakeLegend` on the command line or see the list of the properties of the `nb_graph_ts` in the library.

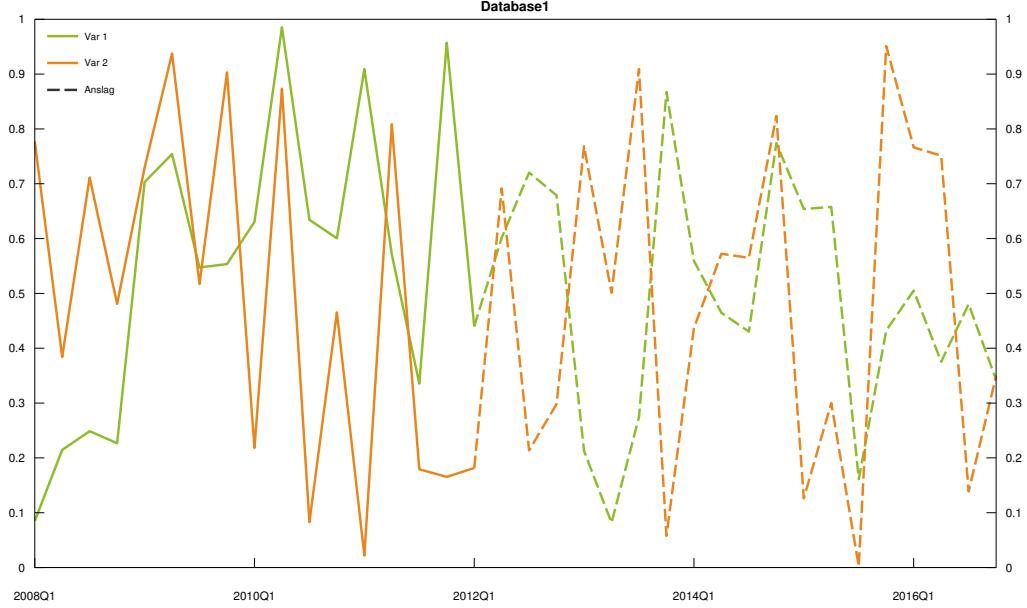


Figure 8: Example of use of the `fakeLegend` property.

6.4.6 Adding labels

It is also possible to add labels to the axes. This could be done with the `xLabel`, `yLabel` and `yLabelRight` properties of the `nb_graph_ts`, the `nb_graph_data` and the `nb_graph_cs` class. Where `xLabel` when set to a string place a label on the x-axis. `yLabel` and `yLabelRight` when set to a string will place a label on the left y-axis and right y-axis respectively. It is also possible to change the font size, font weight and so on with other properties of the `nb_graph_ts`, the `nb_graph_data` and the `nb_graph_cs` class. Please see the library for more on these properties. The example below produces the graph in figure 9.

```

% Adding data to an object of class nb_ts
data      = nb_ts([2,2;1,3;3,2]/1000,'','2012',{'Var1','Var2'});

% Initialize an nb_graph_ts object with the above data
plotter = nb_graph_ts(data);

% Add a x-axis label
plotter.set('xLabel','x-axis');

% Add a y-axis label left
plotter.set('yLabel','y-axis left');

% Add a y-axis label right
plotter.set('yLabelRight','y-axis right');

% Plot the object
plotter.graph

```

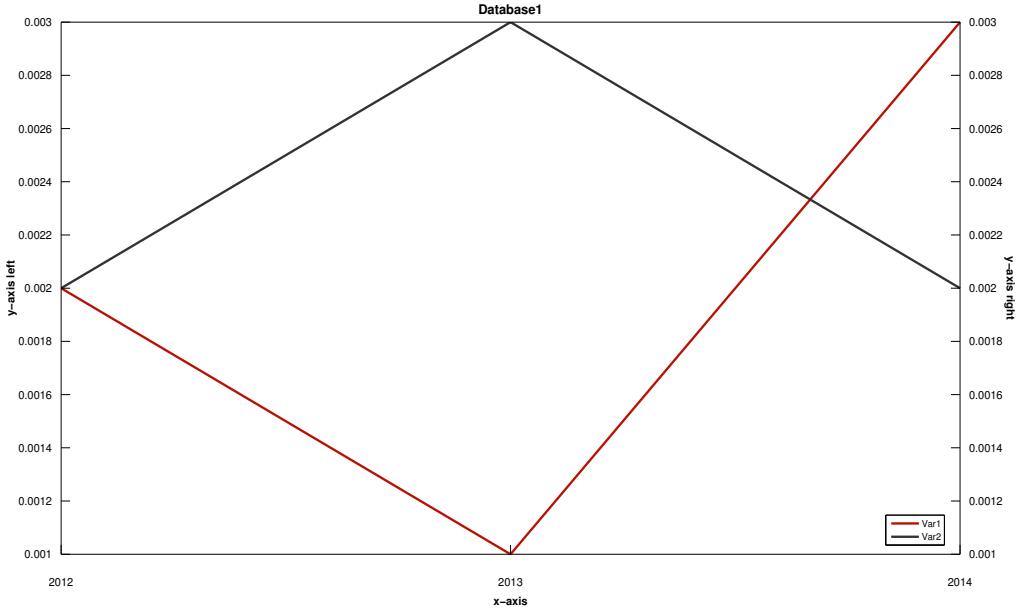


Figure 9: Adding labels to the graph.

6.4.7 Set the graph style

If you are going to produce many graphs and all of them share some of the same settings, the `graphStyle` property of the `nb_graph_ts`, the `nb_graph_data` and the `nb_graph_cs` class can be used.

There are four predefined graph styles for these two classes; 'presentation' and 'presentation_white'. The default graph style used by the `nb_Presentation` class is 'presentation'. Here is an example of the use of the property `graphStyle`:

```
data      = nb_ts([2,2;-1,3;3,-2;3,4;-2,3;1,2],'', '2012Q1',...
                  {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set the graphStyle property to do this ('presentation' is a predefined
% graph style)
plotter.set('graphStyle', 'presentation');
plotter.graph
```

You can also provide your own settings in a saved .m file. Say you saved your settings to a .m file with the name `exampleStyleFile.m`, you can then set the `graphStyle` property to the string '`exampleStyleFile`'. Be aware that the extension must be excluded from the provided input. Here is an example of using an example file:

```
data      = nb_ts([2,2;-1,3;3,-2;3,4;-2,3;1,2],'', '2012Q1',...
                  {'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set the graphStyle property to do this. Now using a .m file.
% Please don't add the extension!
plotter.set('graphStyle', 'exampleStyleFile');
plotter.graph()
```

Where the format of the provided file is the following:

```
% This is an example file you can provide to the nb_graph_ts
% nb_graph_data and nb_graph_cs classes to set the graph style
% Here we are setting properties inside the method call
obj.noTitle          = 1;
obj.shading           = 'grey';
obj.legBox            = 'off';
obj.axesFontSize      = 17;
obj.legFontSize       = 14;
obj.legSpace          = 0.03;
obj.legInterpreter    = 'tex';
obj.yOffset           = 0.03;
obj.dateInterpreter   = 'middle';
```

6.4.8 Adding horizontal lines

To add a horizontal line you can use the horizontalLine property of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class. This must be set to a $1 \times M$ double. Where M will be the number of added horizontal lines. If you want to change the line color of the horizontal line the `horizontalLineColor` can be used. The default color is black. This input must either be a cell array, where each element must be a color name as a string or a 1×3 doubles with the RGB color of each of the added horizontal lines. E.g. `{'black',[0 0 0]}`. In much the same way you can set the horizontal line style with the property `horizontalLineStyle` property. It must be set to a cell array of the line styles of each horizontal line. E.g. `{'-','--'}`. Lastly it is possible to set the width of the horizontal lines width the property `horizontalLineWidth`. This must be set to a scalar. Default is 1. Here is some examples:

```
data    = nb_ts(rand(36,2)*3,'','2008Q1',{'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set some properties of the graph to create a horizontal line
plotter.set('horizontalLine', 2, ...
            'horizontalLineColor', 'black',...
            'horizontalLineStyle', '-',...
            'horizontalLineWidth', 1);

plotter.graph()

% Set some properties of the graph to create more horizontal lines
plotter.set('horizontalLine', [2, 2.5],...
            'horizontalLineColor', {'orange','blue'},...
            'horizontalLineStyle', {'-','-{-}'},...
            'horizontalLineWidth', 1);

plotter.graph()
```

6.4.9 Adding vertical lines

To add a vertical line you can use the `verticalLine` property of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class. This must be set to a $1 \times M$ cell array. Where M will be the number of added vertical lines. Each element must either be a string with the date or the type for where to place the vertical line or a 1×2 cell array of two dates or types for where to place the vertical line. I.e. between the dates or types given. If you want to change the line color of the vertical line the `verticalLineColor` can be used. The default color is black. This input must either be a cell array, where each element must be a color name as a string or a 1×3 doubles with the RGB color of each of the added vertical lines. E.g. `{'black',[0 0 0]}`. In much the same way you can set the vertical line style with the property `verticalLineStyle` property. It must be set to a cell array of the line styles of each vertical line. E.g. `{'-','--'}`. Lastly it is possible to set the width of

the vertical lines width the property verticalLineWidth. This must be set to a scalar. Default is 1. Here is some examples:

```

data      = nb_ts(rand(36,2)*3,'','2008Q1',{'Var1','Var2'});
plotter = nb_graph_ts(data);

% Set some properties of the graph to create a vertical line
plotter.set('verticalLine',      {'2012Q1'},...
            'verticalLineColor', 'black',...
            'verticalLineStyle', '-',...
            'verticalLineWidth', 1);

plotter.graph()

% Set some properties of the graph to create more vertical lines
plotter.set('verticalLine',      {'2012Q1','2014Q1','2014Q2'},...
            'verticalLineColor', {'orange','blue'},...
            'verticalLineStyle', {'-','{-}{-}'},...
            'verticalLineWidth', 1);

plotter.graph()

```

6.4.10 Normalize the font units

It is possible to normalize the font of the text object of the figure. This could be done with the fontUnits property of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` classes. I.e. set it to 'normalized'. This will normalize the current font size properties, i.e. `axesFontSize`, `legFontSize`, `titleFontSize`, `xLabelFontSize` and `yLabelFontSize`, to the size of the font which the former font size would have plotted on the default axes positions and a resolution of the screen of 1680x1050.²⁰

Only set this property with use of the set method of the `nb_graph_ts`, `nb_graph_data` and `nb_graph_cs` class. If not the text will most certainly disappear.

6.4.11 Save the graph(s) to a file

It is also possible to save the produced graphs to a pdf file by setting the property `saveName` of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` object. Other formats are supported as well. Just type in `help nb_graph_cs.fileFormat` in the command line to see the file formats supported. Lastly you can resize the printed graph to cover the full paper size by using setting the `crop` property to 1. Default is not to crop. I.e. the size of the graph printed depend on the size of the figure window shown on the screen. E.g:

```

data = nb_ts('example_ts_quarterly');
plotter = nb_graph_ts(data);
plotter.set('title','test','titleFontSize',14);

% Set the saveName property to save it to a pdf
plotter.set('saveName','simpleFigure','crop',1);

% Plot the object
plotter.graph

```

²⁰Be aware that this does not say that the font will relate to the axes exactly in the same way irrespectively of the position of the axes and the resolution of the screen. The reason is that the font size of the text changes discontinuously, and the font size is only normalized to the height of the axes. This will in any case prevent the text to overlap if you switch between screens with different resolution.

If the graphing method produce more figures each figure will be saved down as separate pdf files. If you want to save all figures down to one pdf, you can set the property pdfBook to 1. (This is only an option when saving down the figures to pdf.)

6.4.12 Writing the data of the graph(s) to Excel

If you want to save the data behind a graph you can utilize the method saveData of the nb_graph_ts, the nb_graph_data or the nb_graph_cs class. Here is an example of a time-series graph:

```
% Reading data from a excel spreadsheet
data = nb_ts('example_ts_quarterly');

% Initialize an nb_graph_ts object with the above data
plotter = nb_graph_ts(data);

% Set one property of the object
plotter.set('title','test');

% Set more properties of the object at one function call
plotter.set('variablesToPlot',{'QUA_RNFOLIO','QUA_RNLC_BKF'},...
            'startGraph','1997Q1');

% Plot the object
plotter.graph

% Save the data of the plot to a excel file
plotter.saveData('test')
```

Be aware that only the data of the plotted variables and their time-span is saved to the excel spreadsheet.

An example of the cross-sectional case:

```
data      = nb_cs([1,2; 2,4; 3,5; 2,1; 3,4; 2,5],[],...
                  {'type1','type2','type3','type4','type5','type6'},...
                  {'Var1','Var2'});
plotter = nb_graph_cs(data);
plotter.set('variablesToPlot',{'Var1'},'typesToPlot',{'type1','type2'});
plotter.graph();

% Save the data of the plot to a excel file
plotter.saveData('test')
```

Again be aware that only the data on the types and variables plotted are saved to the excel spreadsheet.

6.5 nb_graph_ts examples

See the example script file found at \\NBTOOLBOX\Examples\Graphics\nb_graph_tsExamples.m

6.6 nb_graph_cs examples

See the example script file found at \\NBTOOLBOX\Examples\Graphics\nb_graph_csExamples.m

6.7 nb_graph_data examples

See the example script file found at \\NBTOOLBOX\Examples\Graphics\nb_graph_dataExamples.m

6.8 Adding annotations

To add annotation you can utilize the annotation property of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class. This property must be assign an object or a cell array of objects which is of a subclass of the `nb_annotation` class. Here is a list of all the subclasses of the `nb_annotation` class²¹:

- `nb_arrow` Adding arrows
- `nb_barAnnotation` Adding text to bar plots
- `nb_drawLine` Plot a line
- `nb_drawPatch` Plot a patch
- `nb_textArrow` Plotting a arrow with a text box
- `nb_textBox` Adding a text box

So to add an annotation object to a graph you can do as follows (ann, ann1 and ann2 is all objects of one of the above listed classes):

```
% One annotation object
plotter.set('annotation',ann);

% More annotation objects
plotter.set('annotation',{ann1,ann2});
```

It is only possible to add annotation using the `graph` method of the `nb_graph_ts`, the `nb_graph_data` or the `nb_graph_cs` class.

6.8.1 More examples

See the example script file found at \\NBTOOLBOX\\Examples\\Graphics\\nb_annotationExamples.m

6.9 Creating subplots

With the use of the `nb_graph_subplot` class you can combine `nb_graph_ts`, `nb_graph_data` and/or `nb_graph_cs` objects in one figure as different subplots. The below code gives an example:

```
% Create objects of the subplots
data1 = nb_ts([2,2,2;1,3,2;3,2,2],'', '2012', {'Var1', 'Var2', 'Var3'});
plotter1 = nb_graph_ts(data1);
plotter1.set('legType','nb', 'legPosition',[0.5 0.5], 'xLabel', 'x-axis');

data2 = nb_ts([2,2,2;1,3,2;3,2,2],'', '2012', {'Var1', 'Var2', 'Var3'});
plotter2 = nb_graph_ts(data2);

data3 = nb_cs([2,2,2;1,3,2;3,2,2],'', {'type1', 'type2', 'type3'}, ...
             {'Var1', 'Var2', 'Var3'});
plotter3 = nb_graph_cs(data3);

data4 = nb_cs([2,2,2;1,3,2;3,2,2],'', {'type1', 'type2', 'type3'}, ...
             {'Var1', 'Var2', 'Var3'});
plotter4 = nb_graph_cs(data4);

% Initialize the nb_graph_subplot object
plotter = nb_graph_subplot(plotter1, plotter2, plotter3, plotter4);
```

²¹To get more information on how to use these classes see the library.

```
% Graph the figure
plotter.graph();
```

Which will result in figure 10.

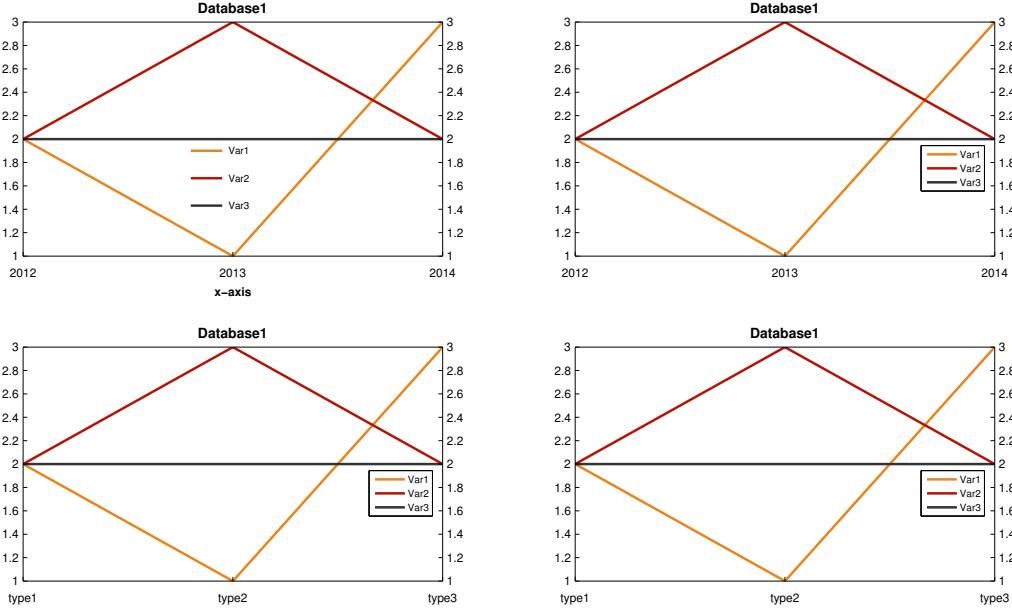


Figure 10: Combing nb_graph_ts and nb_graph_cs objects as subplots in one figure

Also, the `nb_graph_subplot` class has some properties you can set using the `set` method. To find out the properties which are supported, type in `properties(nb_graph_subplot)` on the command line. These properties are mainly font size settings and settings on how to save down the figure to a file. The class also has a method to save down the data of the graph produced to excel. I.e. the `saveData` method.

Objects of this class cannot be included in a graph package, but can be included in a presentation.

6.10 Create a package of graphs

Sometimes its useful to create a package of graphs, so that you have a simple way to produce output in both English and Norwegian. This could be done with the two classes `nb_graph_package` and `nb_graph_adv`. The `nb_graph_adv` is a class for adding figure titles and footers in both English and Norwegian to a `nb_graph_ts`, a `nb_graph_data` or a `nb_graph_cs` object. How this is done will be described first, then it will be described how to incorporate more graphs in one package.

6.10.1 Adding figure title and footer

Sometimes it is easier to first provide an example and then describe what is going on, so:

```
% Reading data from a excel spreadsheet
data = nb_ts('example_ts_quarterly');

% Initialize an nb_graph_ts object with the above data
plotter = nb_graph_ts(data);

plotter.set('graphStyle',      'mpr',...
            'variablesToPlot', {'Var1','Var2'},...
```

```

'legends',           {'FÅ,rste','Andre'},...
'legPosition',      [0.1 0.7],...
'yLim',             [0,1],...
'ySpacing',          1 ...
);

figureTitleNorwegian = {'En tittel til figuren',...
    plotter.timespan('')};

footerNorwegian      = {'1) En fotnote.',...
    'Kilde: En plass.'};

figureTitleEnglish   = {'A title for the graph',...
    plotter.timespan('english')};

footerEnglish         = {'1) A footnote.',...
    'Source: From somewhere.'};

legendsEng           = {'First',...
    'Second'};

plotterAdv = nb_graph_adv(plotter, ...
    'legendsEng', legendsEng, ...
    'figureTitleEng', figureTitleEnglish, ...
    'footerEng', footerEnglish, ...
    'figureTitleNor', figureTitleNorwegian, ...
    'footerNor', footerNorwegian...
);

% Graph Norwegian version
plotterAdv.graphNor()

% Graph English version
plotterAdv.graphEng()

```

As the example shows we must first read the data to a `nb_ts` object, then we initialize the `nb_graph_ts` object. As we want to change how the figure is plotted, we set different properties of the `nb_graph_ts` object `plotter`. Then we are ready to add figure titles and footers in both English and Norwegian. This is done when initializing the `nb_graph_adv` object `plotterAdv` with the inputs given above. Be aware that all the above inputs are cell arrays of strings. For the figure titles and footers (given by the variables `figureTitleNorwegian`, `footerNorwegian`, `figureTitleEnglish` and `footerEnglish`) a new element of the cell will be a new line. Be aware of the square brackets enclosing the two first lines of the variables, this ensures that the two lines of strings are merged into one string. This is the reason that the above code result in only two lines in the figure 11. It is recommended to always let one line of code represent one line of the figure title or footer, but because of space limitation the above example deviated from this. (It is also a good idea to use the comma sign to represent a line break.) The last input above is the translation of the legends to English. This input should always be given if a English version of the figure is produced. **Here you need to be aware that if you use the patch and fakeLegend properties of the nb_graph_ts class you must also provide the legend description for these inputs as well. The translation of the legends for the patch property must be provided first and the translation of the legends provided by the fakeLegend property must be provided last in the legendsEng cell array.**

The `nb_graph_adv` class also have some more properties which affect the excel output file when the `nb_graph_adv` object is included in a graph package. To set the name of the given figure to be included on the index pages in English and Norwegian the properties `figureNameEng` and `figureNameNor` can be used. Both must be set to a string. The `remove` property can be utilized to prevent the data of some variables to be saved to the excel output. This property must be assign a cell array of strings with the names of the variables which are not going to be saved to the excel file. If all variables of the plot is

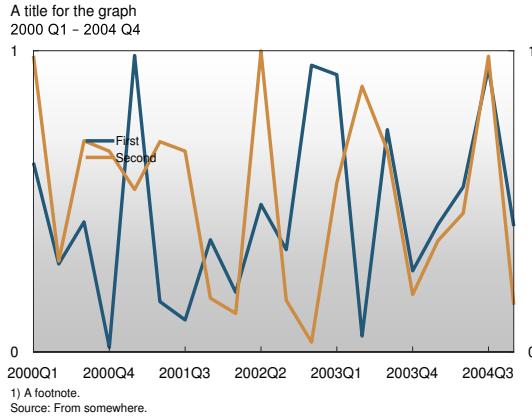


Figure 11: Adding a figure title and footers to an nb_graph_ts object.

being removed, a special message on the worksheet of that figure will be given. I.e. data publiseres ikke/data are not published. Finally, you can use the forecastDate property to color the text of some data to indicate that it is a forecast. It must be assigned as a string with the date the forecast starts (e.g. '2012Q1') or a cellstr with the individual start dates of each variable plotted. (e.g. {'Var1','2012Q1','Var2','2012Q2',...}). If a variable is not given in this cellstr none of the data of that variable will be colored as being forecasts.

The nb_graph_adv class has also some properties which sets the numbering of the figure when included in a graph package. The first is the letter property which when set to 1 will append a letter to the figure number. I.e. Chart 1.1a. And if more graphs which are added to the graph package has this property set to 1 the following graphs will be numbered as Chart 1.1b, 1.1c and so on. The second input is the counter input, this input reset the figure counter. I.e. say you until now have added three graphs, which will mean that next graph you add will normally get the number four. But say you set the property counter to 5 it will of course get the number 5 instead. I.e. Chart 1.5. And the following added graphs will then proceed counting from this number. The last is the jump property which will make the graph counter jump the given number of graphs. E.g. if set to 2 this property will make it jump from Chart 1.1 to Chart 1.3.

6.10.2 Initializing a graph package

To initialize a package you must use the nb_graph_package class. Here is an example:

```
graphNumber = 1;
chapter      = 1;

% Initialize a nb_graph_package object
package = nb_graph_package(graphNumber,chapter);
```

The first input, i.e. graphNumber in the above example, is the starting number of the graph counter. The second input, i.e. chapter int he above example will be the chapter of the graph package. This means in this example the first graph will get the name Chart

1.1 in English and Figur 1.1 in Norwegian.

This class has only one settable property. I.e. lookUpMatrix. The lookUpMatrix property must be set to a string with the name of a .m file on the format (without extension), or a struct:

```
obj.lookUpMatrix = {  
    % mnemonics, English translation Norwegian translation  
    'QUA_VAR1', 'First variable', 'Variabel nummer en'  
    'QUA_VAR2', 'Second variable', 'Variabel nummer to'  
    'QUA_VAR3', 'Third variable', 'Variabel nummer tre'  
    'QUA_VAR4', 'Fourth variable', 'Variabel nummer fire'  
};
```

This property is used to look up the variables names of the data behind the graphs of the graph package and translate them to Norwegian and English. This only applies to the created excel output.

You are now ready to add different graphs to the graph package.

6.10.3 Adding an graph object to the graph package

To add a graph to the graph package you must use the `add` method of the `nb_graph_package` class. Remember that it is only possible to add an `nb_graph_adv` object to the graph package object. So if `plotterAdv` is an object of class `nb_graph_adv` you can use the following code to add the graph to the package object, which is of class `nb_graph_package`:

```
package = package.add(plotterAdv);
```

It is only possible to add one graph object at the time.

6.10.4 Print the graph(s) to a pdf file

When all the graphs are added to the graph package you can use the method `writePDF` to print the figures produced to a pdf file. This method takes two inputs. The first input will be the save name of the pdf file. It must be given as string. The second input must be the language to use. Either 'norsk' or 'english'. An example will follow:

```
% PDF in norwegian  
package.writePDF('test_nor','norsk');  
  
% PDF in english  
package.writePDF('test_eng','english');
```

6.10.5 Writing data of the graph package to Excel

To save the data behind the figures to excel you can use the `saveData` method of the `nb_graph_package` class. This method takes five inputs. The first input will be the name of the saved file, as a string. The second input must be either 'norsk', 'english' or 'both'. This input sets the format of the saved excel spreadsheet, i.e. if should be given in Norwegian, English or in both languages respectively. The third input sets the heading of the Norwegian index page, This must either be a 1x1 cellstr or a 1x2 cellstr. E.g:

```
% A 1x1 cellstr  
firstPageNameNor = {'Et eksempel'};  
  
% A 1x2 cellstr  
firstPageNameNor = {'Et eksempel',...  
    'med masse null'};
```

The fourth input must be the same as the second input only translated to English. I.e. the heading of the English index page. The fifth input will if set to 1 round-off all the save data of the excel spreadsheet to two decimals. Here is an example:

```
firstPageNameNor = {'Et eksempel',...
    'med masse null'};
firstPageNameEng = {'An example',...
    'with a lot of crap'};
roundoff         = 1;

package.saveData('test_excel_output','both',firstPageNameNor,...
    firstPageNameEng,roundoff);
```

Be aware that you must run the writePDF command before the saveData command if default fans are added to a graph of the package.

6.10.6 Full example script

See the example script file found at \\NBTOOLBOX\\Examples\\Graphics\\nb_graph_packageExample.m

6.11 Basic plot classes and functions

All the above classes are calling some basic plot classes and functions under the hood. These are in a sense more general, but are also harder to use. They are also limited in that they only support data as doubles, as the MATLAB plotting functions also do. Here is a list of these classes:²²

²²Follow the links to get more information on the given class

- `nb_area` Area plot
- `nb_axes` Creating a axes to plot on
- `nb_bar` Bar plot
- `nb_candle` Candle plot
- `nb_fanChart` Create a fan chart
- `nb_fanLegend` Create a legend to the fan chart
- `nb_figure` Creating a figure handle add axes to
- `nb_figureTitle` Adding a figure title to the figure
- `nb_footer` Adding footers to the figure
- `nb_graphPanel` Creating a figure handle add axes to
(Fixes aspect ratio)
- `nb_hbar` Horizontal bar plot
- `nb_highlight` Highlighted areas which spans the whole y-axis
- `nb_horizontalLine` A horizontal line which spans the whole x-axis
- `nb_legend` Adding a legend to the plot
- `nb_line` Plotting a line
- `nb_patch` Plotting a patch
- `nb_pie` Pie plot
- `nb_plot` Plotting more lines
- `nb_plotComb` Combine different plot types in one plot
- `nb_radar` Radar plot
- `nb_scatter` Scatter plot
- `nb_title` Adding a title to the plot
- `nb_verticalLine` Adding a vertical line which spans the whole y-axis
- `nb_xlabel` Adding a x-axis label
- `nb_ylabel` Adding a y-axes label

Here is a list of the functions:²³

- `nb_subplot` Create axes in tiled positions
- `nb_subplot_position` Get the position of the tiled axes
- `nb_subplotSpecial` Create axes in tiled positions in a special way

²³Follow the links to get more information on the given function

6.11.1 Examples

See the example script file found at \\NBTOOLBOX\\Examples\\Graphics\\basicPlotClassesExamples.m

7 Basic Econometrics

Throughout this chapter, the data variable will refer to the following time series:

```
% Create time series data
rng(1); % Set seed

draws = randn(100,1);
sim1 = filter(1,[1,-0.8],draws);
sim2 = filter([1,-0.8],1,draws);
sim = [sim1,sim2];

data = nb_ts(sim,'','2012Q1',{'AR','MA'});
```

7.1 Autocorrelation

To compute the autocorrelation of a time series, use the `autocorr` method of the `nb_ts` class:

```
acf = autocorr(data, 5); % 5 lags
acf.plot('graphSubPlots');
```

To calculate confidence bounds, supply the desired calculation method and significance value:

```
acf = autocorr(data, 5, 'asymptotic', 0.05);
acf.plot('graphSubPlots');
```

To compute the partial autocorrelation of a time series, use the `parcorr` method of the `nb_ts` class instead.

7.2 Unit root tests

To test a time series for unit root, use the `nb_unitRootTest` class.

```
test = nb_unitRootTest(data, 'adf'); % or 'pp'.
print(test)
```

For information about optional inputs, look at the documentation for `nb_adf` and `nb_phillipsPeron`.

7.3 Cointegration tests

To test time series for cointegration, use the `nb_cointTest` class.

```
% Engle-Granger
test = nb_cointTest(data, 'eg', 'lagLengthCrit', 'aic');
print(test)

% Johansen
test = nb_cointTest(data, 'jo', 'nLags', 4);
```

```
print(test)
```

For information about optional inputs, look at the documentation for `nb_egoint` and `nb_jcoint`.

7.4 Information Criterion

The different information criterion are calculated using the `nb_infoCriterion` function. For more see the help for this function in MATLAB.

7.5 Frequency zero spectrum estimation

Frequency zero spectrum estimation is done by the `nb_zeroSpectrumEstimation` function. For more see the help for this function in MATLAB.

7.6 Kernel functions

See `nb_bartlettKernel`, `nb_parzenKernel` and `nb_quadraticSpectralKernel`. For more see the help for these functions in MATLAB.

7.7 Bootstrapping

How to bootstrap a model may be very model dependent, but the bootstrapping of residuals is done with the function `nb_bootstrap`. For more see the help for this function in MATLAB.

8 Single equation, time-series

To estimate a single equation model, use the `nb_singleEq` class.

To initialize a `nb_singleEq` object, start by calling the `nb_singleEq.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_singleEq` constructor:

```
% Create some data
EXO1 = randn(100,1);
EXO2 = randn(100,1);
DEP = 0.2*EXO1 + -0.5*EXO2 + randn(100,1);
data = nb_ts([DEP,EXO1,EXO2],'', '1990Q1', {'Dep','Exo1','Exo2'});

% Formulate model
options = nb_singleEq.template();
options.data = data;
options.dependent = {'Dep'};
options.exogenous = {'Exo1', 'Exo2'};
options.nLags = 4;

% Initialize model object
model = nb_singleEq(options);
```

For information about the different options, use the `nb_singleEq.help` method:

```
nb_singleEq.help() % All options
nb_singleEq.help('data') % Specific option
```

To estimate the model, use the `estimate` method:

```

model = estimate(model);
print(model)

```

Where the last statement prints the estimation results to the command line.

8.1 OLS

To estimate the model using OLS, set the 'estim_method' property to 'ols'.

8.2 TSLS

To estimate the model using TSLS, set the 'estim_method' property to 'tsls' and supply endogenous and instrument variables as shown below:

```

% Create some artificial data
sigma = [0.4, 0.1;
          0.1, 0.6];
S      = nb_mvnrnd(100,1,[0,0],sigma);
EXO   = randn(100,1); % Uncorrelated with RES, but correlated with ENDO
ENDO  = S(:,1) + EXO;
RES   = S(:,1);
DEP   = 0.5*ENDO + RES + randn(100,1)*0.5;

data = nb_ts([DEP,ENDO,EXO],'', '1980Q1', {'Dep','Endo','Exo'});

% Formulate model
dep    = {'Dep'};
endo  = {'Endo'};
instr  = {'Endo', {'Exo'}};
modelTS = nb_singleEq...
    'estim_method', 'tsls',...
    'data',        data,...,
    'dependent',  dep,...,
    'endogenous', endo,...,
    'instruments', instr);

% Estimate
modelTS = estimate(modelTS);
print(modelTS)

```

8.3 Estimation results

See corresponding section of [Paulsen \(2021\)](#).

8.4 Automatic model selection

For automatic lag length selection, set the 'modelSelection' property:

```

options.modelSelection      = 'lagLength';
% or 'autometrics'
options.criterion          = 'aic';
options.modelSelectionAlpha = 0.05;
options.maxLagLength       = 10;

```

To indicate which regressors to keep fixed during automatic model selection, set the 'modelSelectionFixed' property to a logical vector:

```

options.modelSelectionFixed = [true, false];

```

The algorithm of the automatic model selection can be found in `nb_automaticModelSelection`. Most models either supports 'lagLength' or 'autometrics'.

8.5 Test statistics

8.5.1 ARCH (Autoregressive conditional heteroskedasticity)

To test for ARCH, use the `nb_archTestStatistic` class:

```
test = nb_archTestStatistic(model, 'nLags', 4);
print(test)
```

For all available options, run `nb_archTestStatistic.template()`.

8.5.2 Autocorrelation

To test for autocorrelation, use the `nb_autocorrTestStatistic` class:

```
test = nb_autocorrTestStatistic(model, 'nLags', 4);
print(test)
```

For all available options, run `nb_autocorrTestStatistic.template()`.

8.5.3 Breusch-Pagan test for heteroskedasticity

To test for heteroskedasticity, use the `nb_breuschPaganTestStatistic` class:

```
test = nb_breuschPaganTestStatistic(model);
print(test)
```

For all available options, run `nb_breuschPaganTestStatistic.template()`.

8.5.4 Chow test for a structural break

To test for a structural break, use the `nb_chowTestStatistic` class:

```
test = nb_chowTestStatistic(model, 'breakpoint', '1970Q1');
print(test)
```

For all available options, run `nb_chowTestStatistic.template()`.

8.5.5 Durbin-Wu-Hausman test for exogeneity

To test for exogeneity, use the `nb_durbinWuHausmanStatistic` class:

```
test = nb_durbinWuHausmanStatistic(model, 'endogenous', {'Var1'}, ...
    'instruments', {'Instr1','Instr2'});
print(test);
```

For all available options, run `nb_durbinWuHausmanStatistic.template()`.

8.5.6 F-test

To perform a F-test, use the `nb_fTestStatistic` class:

```
% Restrictions
A = [1, 0; 0, 1];
c = [1; 0];

test = nb_fTestStatistic(model, 'A', A, 'c', c);
print(test)
```

For all available options, run `nb_fTestStatistic.template()`.

9 Expression model

This model class uses just-in-time evaluation of expression for estimation and forecasting the model. To estimate a expression model, use the `nb_exprModel` class.

To initialize a `nb_exprModel` object, start by calling the `nb_exprModel.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_exprModel` constructor:

```
t          = nb_exprModel.template();
t.constant = true;
t.data     = data;
t.dependent = {'growth(y(t))'};
t.exogenous = {'growth(y(t-1))', 'growth(x(t-1))'};
t.recursive_estim = true;
model      = nb_exprModel(t);

% Initialize model object
model = nb_singleEq(options);
```

Here `data` is a `nb_ts` object storing at least the two time-series y and x . For information about the different options, use the `nb_exprModel.help` method:

```
nb_exprModel.help() % All options
nb_exprModel.help('data') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line. For more examples go see `NBTOOLBOX\Examples\Econometrics\ExprModel`. Where we also have included an example on how to set up a VAR in this way.

9.1 OLS

To estimate the model using OLS, set the `'estim_method'` property to `'ols'`.

10 Step ahead model

To estimate a step ahead model, use the `nb_sa` class.

To initialize a `nb_sa` object, start by calling the `nb_sa.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_sa` constructor:

```
% Simulate some data
% y_h = a + b*x + e
```

```

y      = nan(100,2);
x1    = nan(100,1);
x2    = nan(100,1);
x1(1) = 1;
x2(1) = 2;
for t = 2:100
    x1(t) = 0.5*x1(t-1) + randn;
    x2(t) = 0.5*x2(t-1) + randn;
end
y(2:100,1) = 0.2 + 0.6*x1(1:99) + 0.4*x2(1:99) + randn(99,1);
y(2:100,2) = 0.3 + 0.1*x1(1:99) + 0.8*x2(1:99) + randn(99,1);
data      = nb_ts([y,x1,x2],'', '1999Q1', {'y', 'y2', 'x1', 'x2'});

% Formulate model
t          = nb_sa.template;
t.data     = data;
t.dependent = {'y'};
t.exogenous = {'x1', 'x2'};
t.constant = 1;
t.nStep    = 4;

% Initialize model object
model = nb_sa(t);

```

As this model is constructed for direct forecast you need to set the nStep options to the number of periods you want to forecast. It is not possible to forecast longer than this option when using the forecast method later on.

For information about the different options, use the `nb_sa.help` method:

```

nb_sa.help() % All options
nb_sa.help('nStep') % Specific option

```

To estimate the model, use the `estimate` method:

```

model = estimate(model);
print(model)

```

Where the last statement prints the estimation results to the command line.

11 ARIMA

To estimate an ARIMA(p,i,q) model, use the `nb_arima` class. Where p is the number of AR terms, i is the level of integration and q is the number of MA terms. Let's first start by generating some artificial data:

```

draws = randn(100,1);
% ARIMA(1,0,0)
sim1 = filter(1,[1,-0.5],draws);
% ARIMA(0,0,1)
sim2 = filter([1,-0.5],1,draws);
% ARIMA(1,0,1)
sim3 = filter([1,-0.5],[1,-0.2],draws);
% ARIMA(1,0,2)
sim4 = filter([1,-0.5,0.2],[1,-0.2],draws);

data  = nb_ts([sim1,sim2,sim3,sim4],'', '2000',...
             {'Sim1','Sim2','Sim3','Sim4'});
dataUR = undiff(data('Sim1'),100,1);
dataUR = rename(dataUR,'variables','Sim1','Sim5');
data  = merge(data,dataUR);

```

So from now on the *data* variable refers to this generated data set. There are two ways to estimate an ARIMA(p,i,q) in NBTOOLBOX, and how to do this is shown in the next two subsections. Please also see section 21 for how to forecast an ARIMA(p,i,q) model.

There are also two ways to initialize a `nb_arima` object. The first way is to use `propertyName`, `PropertyValue` combinations directly to the structure of the `nb_arima` class:

```
model = nb_arima(...  
    'algorithm',      'ml', ...  
    'constant',       0, ...  
    'data',           data, ...  
    'dependent',     {'Sim3'}, ...  
    'AR',             1, ...  
    'MA',             3, ...  
    'integration',   0, ...  
    'estim_start_date', '' , ...  
    'estim_end_date', data.endDate - 11, ...  
    'recursive_estim', 1);
```

Or you can use the static method `nb_arima.template` of the `nb_arima` class to get a struct with the default options:

```
options          = nb_arima.template();  
options.algorithm = 'ml';  
options.constant = 0;  
options.data     = data;  
options.dependent = {'Sim3'};  
options.AR        = 1;  
options.MA        = 3;  
options.integration = 0;  
options.estim_start_date = '';  
options.estim_end_date   = data.endDate - 11;  
options.recursive_estim = 1;  
model            = nb_arima(options);
```

Both will return a `nb_arima` with the same options. To get help on the different options of the `nb_arima` class use the `nb_arima.help` method:

```
h = nb_arima.help('AR')
```

To estimate the model you must use the `estimate` method:

```
model = estimate(model);  
print(model)
```

Where the last statement prints the estimation results to the command line.

11.1 Maximum likelihood

To estimate an ARIMA(p,i,w) model by maximum likelihood set the 'algorithm' option to 'ml'. For more see [Paulsen \(2021\)](#).

11.2 Hannan-Rissan

To estimate an ARIMA(p,i,w) model by the Hannan-Rissan algorithm set the 'algorithm' option to 'hr'. For more see [Paulsen \(2021\)](#).

12 Error-correction model (ECM)

To estimate an ECM, use the `nb_ecm` class. Let's first start by generating some artificial data:

```
% Simulate some data
z      = nan(100,1);
scale = 10;
z(1)  = scale*rand;
for ii = 2:100
    z(ii) = z(ii-1) + scale*rand;
end
x1    = z + randn(100,1);
x2    = z + randn(100,1);
y     = 0.2 + 0.6*x1 + 0.4*x2 + 2*randn(100,1);
w1    = randn(100,1);
w2    = randn(100,1);
data = nb_ts([y,x1,x2,w1,w2],'', '1990Q1', {'y','x1','x2','w1','w2'});
```

To initialize a `nb_ecm` object, start by calling the `nb_ecm.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_ecm` constructor:

```
% Formulate model
t        = nb_ecm.template;
t.data   = data;
t.dependent = {'y'};
t.endogenous = {'x1','x2'};
t.exogenous = {'w1','w2'};
t.constant = 1;

% Initialize model object
model = nb_ecm(t);
```

For information about the different options, use the `nb_ecm.help` method:

```
nb_ecm.help() % All options
nb_ecm.help('constant') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line.

13 VAR

To estimate a VAR model, use the `nb_var` class.

To initialize a `nb_var` object, start by calling the `nb_var.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_var` constructor:

```
% Generate artificial data
obs     = 100;
lambda  = [0.5, 0.1, 0.3, 0.2, 0.2, -0.1;
           0.5, -0.1, -0.2, 0, 0.1, -0.2;
           0.6, -0.2, 0.1, 0, 0.4, -0.1];
rho     = [1; 1; 1];
```

```

sim      = nb_ts.simulate('2012M1',obs,...
{ 'VAR1','VAR2','VAR3'},1,lambda,rho);

% Formulate model
options          = nb_var.template();
options.data     = data;
options.dependent = { 'VAR1','VAR2','VAR3'};
options.nLags    = 2;
options.constant = 0;

% Initialize model object
model = nb_var(options);

```

For information about the different options, use the `nb_var.help` method:

```

nb_var.help() % All options
nb_var.help('constant') % Specific option

```

To estimate the model, use the `estimate` method:

```

model = estimate(model);
print(model)

```

Where the last statement prints the estimation results to the command line.

13.1 OLS

To estimate the model using OLS, set the 'estim_method' option to 'ols'.

13.2 Ridge

To estimate the model using ridge regression, set the 'estim_method' option to 'ridge'.

13.3 LASSO

To estimate the model using LASSO, set the 'estim_method' option to 'lasso'.

13.4 B-VAR

To estimate the model using Bayesian methods and assign prior probabilities to the parameters, set the 'prior' property of the model object. Suitable values for this property are given by the `nb_var.priorTemplate` method.

```
options.prior = nb_var.priorTemplate('jeffrey');
```

Alternatively, using the `setPrior` method on an existing model object:

```
model = setPrior(model, nb_var.priorTemplate('jeffrey'));
```

For all available prior templates, see the documentation for `nb_var.priorTemplate`.

13.5 Empirical baysian

To estimate the model using Empirical Bayesian methods and assign prior probabilities to the parameters, set the 'prior' property of the model object. Suitable values for this property are given by the `nb_var.priorTemplate` method. You also need to set the 'empirical' option to trye

```
options.prior      = nb_var.priorTemplate('glp');
options.empirical = true;
```

Alternatively, using the `setPrior` method on an existing model object:

```
model = setPrior(model, nb_var.priorTemplate('glp'));
model = set(model, 'empirical',true);
```

Only supported for the priors 'glp', 'dsge' and 'nwishart'.

13.6 Time varying parameter bayesian estimation

To estimate the model using a time-varying parameter Bayesian method, set the 'prior' property of the model object as follows

```
options.prior      = nb_var.priorTemplate('kkse');
```

Alternatively, using the `setPrior` method on an existing model object:

```
model = setPrior(model, nb_var.priorTemplate('kkse'));
```

13.7 Identification

To identify a VAR model, use the the `set_identification` method to set Cholesky restrictions or combinations of sign and zero restrictions.

Cholesky:

```
model = set_identification(model, 'cholesky', 'ordering',...
    {'VAR1','VAR2','VAR3'});
```

Combinations:

```
restrictions = {
% Variable, shock, period, type, magnitude
    'VAR1' , 'E_X',      0,    0, []
    'VAR2' , 'E_X',      0,   '- ', []
    'VAR3' , 'E_X',      0,   '>', 0
};

model = set_identification(model, 'combination', 'restrictions', ...
    restrictions, 'maxDraws', 100000);
```

Here we have added a zero restriction on the 'E_X' shocks impact (period=0) on the variable 'VAR1', a minus restriction on the 'E_X' shocks impact (period=0) on the variable 'VAR2' and a magnitude restriction on the 'E_X' shocks impact (period=0) on the variable 'VAR3'.

Variable

The name of the variable that the restriction should apply to, as a string.

Shock

The name of the shock to be identified, as a string.

Period

The period the restriction should apply. Can also be inf, i.e. cumulative long term restriction. Must be a scalar integer.

Type

The type column can include 0, '+', '-', '>' and '<'.

Magnitude

When type is set to '>' or '<' this will be the restriction the shock impact must satisfy to be accepted.

To get the identified residual from an estimated model, use the `getIdentifiedResidual` method:

```
model = solve(model); % First solve the model!
residual = getIdentifiedResidual(model);
```

If the model is underidentified, and you have set the 'draws' option to a number greater than 1, the residual output will be a multi-paged `nb_ts` object with the distributions of the identified residuals. I.e. the identified residuals for each accepted draw.

13.8 Test the VAR

13.8.1 Test for stationarity

To test a VAR model for stationarity, use the `nb_model_generic.getRoots` method to calculate the roots of the companion form of the model.

```
model = solve(model); % Model must be solved (companion form)
[roots, plotter] = getRoots(model);
disp(roots);
plotter.graph();
```

13.8.2 LM-test for autocorrelation

To test for autocorrelation, use the `nb_LMVARTestStatistic` class:

```
test = nb_LMVARTestStatistic(model, 'lag', 1);
print(test)
```

For all available options, run `nb_LMVARTestStatistic.template()`.

13.8.3 Ljung-Box test for autocorrelation

To test for autocorrelation, use the `nb_ljungBoxTestStatistic` class:

```
test = nb_ljungBoxTestStatistic(model);
print(test)
```

For all available options, run `nb_ljungBoxTestStatistic.template()`.

13.8.4 Granger causality

To test for Granger causality, use the `nb_var.grangerCausalityTest` class:

```
[test, pValue, result] = nb_grangerCausalityTest(model);
disp(result)
```

14 VAR with missing observations

In this example we use a dataset stored in a `nb_ts` object data, which is generated from

```
obs      = 100;
lambda   = [0.5, 0.1, 0.3, 0.2, 0.2, -0.1;
            0.5, -0.1, -0.2, 0, 0.1, -0.2;
            0.6, -0.2, 0.1, 0, 0.4, -0.1];
rho      = [1; 1; 1];
dataQ    = nb_ts.simulate('1990Q1', obs, ...
    {'VAR1', 'VAR2', 'VAR3'}, 1, lambda, rho);
dataA    = convert(dataQ, 1, 'diffAverage');
dataAQ   = convert(dataA, 4, '', 'interpolateDate', 'end');
dataAQ   = addPrefix(dataAQ, 'A_');
data     = [dataAQ, dataQ];

% Make some missing observations
dataM = setValue(data, 'VAR1', nan, data.endDate, data.endDate);
```

The data object then stores the variables 'VAR1' , 'VAR2' and 'VAR3' on quarterly frequency, and 'A_VAR1', 'A_VAR2' and 'A_VAR3' on yearly frequency.

14.1 MO-VAR

If you have some missing observations in your VAR you can either use the methods described in 18, or you can use maximum likelihood estimator or some of the B-VAR priors that handle missing observations. In any case you need to use the `nb_var` class.

14.1.1 Maximum likelihood

To do maximum likelihood estimation with (or without) missing data use

```
% Options
t          = nb_var.template();
t.data     = dataM;
t.estim_method = 'ml';
t.dependent = {'VAR1', 'VAR2', 'VAR3'};
t.constant = true;
t.nLags   = 2;
t.doTests = 1;
t.optimizer = 'fmincon';
t.covrepair = true;

% Create model and estimate
model = nb_var(t);
model = estimate(model);
print(model)
```

For information about the different options, use the `nb_var.help` method:

```
nb_var.help() % All options
nb_var.help('estim_method') % Specific option
```

To get the smoothed point estimate of the missing observation use

```
smoothed = getFiltered(model);
```

To draw from the distribution of the parameters you need to set the 'method' input to 'asymptotic'. So when for example calling `parameterDraws` method

```
param = parameterDraws(models,1000,'asymptotic');
```

14.1.2 Bayesian

To do bayesian estimation with missing data use the prior types ending with MF, e.g.

```
% Prior
prior      = nb_var.priorTemplate('minnesotaMF');
prior.method = 'mci';

% Options
t          = nb_var.template();
t.data     = dataM;
t.dependent = {'VAR1','VAR2','VAR3'};
t.constant = true;
t.nLags    = 2;
t.prior    = prior;
t.draws    = 500;

% Create model and estimate
model = nb_var(t);
model = estimate(model);
print(model)
```

To get the smoothed point estimate of the missing observation use

```
smoothed = getFiltered(model);
```

14.2 MF-VAR

If you want to estimate a mixed frequency VAR, you need to use the `nb_var` class. Both maximum likelihood and Bayesian estimation methods are supported.

14.2.1 Maximum likelihood

To do maximum likelihood estimation of a mixed frequency VAR with (or without) missing data use

```
% Options
t          = nb_mfvar.template();
t.data     = data;
t.estim_method = 'ml';
t.dependent = {'VAR1','A_VAR2'};
t.frequency = {'A_VAR2',1};
t.mapping   = {'A_VAR2','diffAverage'}; % Approximation
t.constant  = true;
t.nLags    = 2;
t.optimizer = 'fmincon';
t.covrepair = true;

% Create model and estimate
model = nb_mfvar(t);
model = estimate(model);
print(model)
```

For information about the different options, use the `nb_mfvar.help` method:

```
nb_mfvar.help() % All options
nb_mfvar.help('estim_method') % Specific option
```

To get the smoothed point estimate of the high frequency and missing observations use

```
smoothed = getFiltered(model);
```

To draw from the distribution of the parameters you need to set the 'method' input to 'asymptotic'. So when for example calling `parameterDraws` method

```
param = parameterDraws(models,1000,'asymptotic');
```

14.2.2 Bayesian

To do bayesian estimation of a mixed frequency VAR use the prior types ending with MF, e.g.

```
% Prior
prior      = nb_mfvar.priorTemplate('minnesotaMF');
prior.method = 'gibbs';
prior.S_scale = 0.00001;

% Options
t          = nb_mfvar.template();
t.data     = data;
t.dependent = {'VAR1','A_VAR2'};
t.frequency = {'A_VAR2',1};
t.mapping   = {'A_VAR2','diffAverage'}; % Approximation
t.constant  = true;
t.nLags    = 2;
t.prior    = prior;
t.draws    = 500;

% Create model and estimate
model = nb_mfvar(t);
model = estimate(model);
print(model)
```

To get the smoothed point estimate of the high frequency and missing observations use

```
smoothed = getFiltered(model);
```

14.3 Time varying parameter bayesian estimation

See the example; NBTOOLBOX\Examples\Econometrics\MixedFrequency\test_nb_mfvar_timeVarying.m.

14.4 Variables observed at different frequencies

14.4.1 Change in frequency

See the example; NBTOOLBOX\Examples\Econometrics\MixedFrequency\test_nb_mfvar_changeFreq.m.

14.4.2 Observed at different frequencies at the same time

See the example; NBTOOLBOX\Examples\Econometrics\MixedFrequency\test_nb_mfvar_combineFreq.m.

14.5 Adding measurement restrictions

See the example; NBTOOLBOX\Examples\Econometrics\VAR\nb_var_measEqRest.m.

15 MIDAS

The data in this example is the same as was the case in section 14. To estimate a single equation MIDAS model, use the `nb_midas` class.

To initialize a `nb_midas` object, start by calling the `nb_midas.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_midas` constructor:

```
% Formulate model
t          = nb_midas.template();
t.data      = data;
t.algorithm = 'unrestricted';
t.dependent = {'A_VAR1'};
t.exogenous = {'VAR2'};
t.constant  = true;
t.frequency = 1;
t.nStep     = 4;
t.nLags     = 4;
t.doTests   = 1;
t.AR        = 0;
t.draws     = 500;

% Initialize model object
model = nb_midas(t);
```

For information about the different options, use the `nb_midas.help` method:

```
nb_midas.help() % All options
nb_midas.help('constant') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print_estimation_results(model)
```

Where the last statement prints the estimation results to the command line.

If you want to forecast the model you can follow the same steps as explained in section 21.

16 Factor models using principal component

The dataset (data), which is an object of class `nb_ts`, must in this example include the following variables:

```
dep = {'QSA_PCPI', 'QSA_PCPIJAE'};
obs = {'QSA(CG)', 'QSA(CP)', 'QSA(JC)', 'QSA(JG)', 'QSA(JH)', 'QSA(JOS)', ...
       'QSA(IMN)', 'QSA(WILMN)', 'QSA(XMN)', 'QSA(MMN)', 'QSA(XS)', 'QSA(Y)'},
```

16.1 Principal component

The `pca` method of the `nb_ts` class may be used to do principal component analysis.

```
dataPCA    = window(data, '', '', obs);
[F, lambda] = pca(dataPCA, 2);
```

Where the output F and lambda are the estimated factors and the factor loadings respectively. For more see help on the method `nb_ts.pca`.

16.2 Single equation factor model

To estimate a single equation factor model, use the `nb_fm` class.

To initialize a `nb_fm` object, start by calling the `nb_fm.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_fm` constructor:

```
% Formulate model
t = nb_fm.template();
t.dependent = dep;
t.observables = obs;
t.data = data;
t.nFactors = 4;

% Initialize model object
model = nb_fm(t);
```

For information about the different options, use the `nb_fm.help` method:

```
nb_fm.help() % All options
nb_fm.help('constant') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line.

16.3 Step ahead factor model

To estimate a step ahead factor model, use the `nb_fmsa` class. This model class can be used to produce direct forecast using factors.

To initialize a `nb_fmsa` object, start by calling the `nb_fmsa.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_fmsa` constructor:

```
% Formulate model
t = nb_fmsa.template();
t.dependent = dep;
t.observables = obs;
t.data = data;
t.nFactors = 4;
t.nStep = 8;

% Initialize model object
model = nb_fmsa(t);
```

As this model is constructed for direct forecast you need to set the `nStep` options to the number of periods you want to forecast. It is not possible to forecast longer than this option when using the `forecast` method later on.

For information about the different options, use the `nb_fmsa.help` method:

```
nb_fmsa.help() % All options
nb_fmsa.help('constant') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line.

16.4 FA-VAR

To estimate a FA-VAR model, use the `nb_favar` class.

To initialize a `nb_favar` object, start by calling the `nb_favar.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_favar` constructor:

```
% Formulate model
t = nb_favar.template();
t.dependent = dep;
t.observables = obs;
t.data = data;
t.nLags = 2;
t.nFactors = 4;
t.nStep = 8;

% Initialize model object
model = nb_favar(t);
```

For information about the different options, use the `nb_favar.help` method:

```
nb_favar.help() % All options
nb_favar.help('constant') % Specific option
```

To estimate the model, use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line.

17 Dynamic factor model

To estimate a dynamic factor model, use the `nb_fmdyn` class.

To initialize a `nb_fmdyn` object, start by calling the `nb_fmdyn.template()` method. This will return a struct with the default model options. Modify this structure as desired before passing it to the `nb_fmdyn` constructor:

```
% Formulate model
t = nb_fmdyn.template();
t.data = data;
t.nLags = 1;
t.observables = data.variables;
t.transformation = 'standardize';

% Initialize model object
model = nb_fmdyn(t);
```

Here we assume that the data on the observed variables of the model are collected in the `nb_ts` object `data`. For information about the different options, use the `nb_fmdyn.help` method:

```
nb_fmdyn.help() % All options
nb_fmdyn.help('constant') % Specific option
```

17.1 Expected maximum likelihood

To estimate the model using expected maximum likelihood you need to set the 'estim_method' to 'dfmeml':

```
model = set(model, 'estim_method', 'dfmeml');
```

To execute the estimation use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line. A full example can be found at NBTOOLBOX\Examples\Econometrics\FactorModels\Non_stationary. Where we also may allow for a non-stationary dynamic factor model.

17.2 Time varying parameter bayesian estimation

To estimate the model using a baysian method you need to set the 'prior' option:

```
prior = nb_fmdyn.priorTemplate();
model = setPrior(model,prior);
```

To execute the estimation use the `estimate` method:

```
model = estimate(model);
print(model)
```

Where the last statement prints the estimation results to the command line. A full example can be found at NBTOOLBOX\Examples\Econometrics\FactorModels\example_dynFactorModel.m.

18 Missing observations

If there are missing observations in the data, this can be dealt with when estimating VAR-models. We therefore build on the example which was presented in section 14. To initialize a `nb_var` object, start by calling the `nb_var.template()` method:

```
% Formulate model
options          = nb_var.template();
options.data     = dataM; % Data with missing obs
options.dependent = {'Var1', 'Var2', 'Var3'};
options.nLags    = 2;
options.constant = 0;
options.missingMethod = 'forecast';

% Initialize model object
model = nb_var(options);

% Estimate the model
model = estimate(model);
print(model)
```

For information about the options of missing observations, use the `nb_missingEstimator.help` method:

```
nb_missingEstimator.help() % All options
```

Forecasting can be done as documented in section 21, but now also nowcast on the missing observations are produced.

19 DSGE

DSGE models can be solved with use of the `nb_dsge` class. See the corresponding section of the DAG documentation for more on the algorithm used to solve DSGE models.

19.1 Standard 1st order solution

First you need to write down the model in a .nb file. See the DSGE->Model file section in the DAG documentation for more on the model file syntax. In this example we use a simplified version of the Ramirez, Waggoner and Zha (2012) model. The model file is in this case

```
endogenous
PAI,Y,R,RR

exogenous
EPS_R

parameters
betta,eta,kappa,rhor,sigr,mu,psi

model
  1=betta*(1-.5*kappa*(PAI-1)^2)*Y*R(+1)/((1-
  .5*kappa*(PAI(+1)-1)^2)*Y(+1)*exp(mu)*PAI(+1));
  1-eta+eta*(1-.5*kappa*(PAI-1)^2)*Y+betta*kappa*(1-.5*kappa*(PAI-1)^2)*
  (PAI(+1)-1)*PAI(+1)/(1-.5*kappa*(PAI(+1)-1)^2)-kappa*(PAI-1)*PAI;
  (R(-1)/steady_state(R))^rhor*(PAI/steady_state(PAI))^{((1-rhor)*psi)}*
  exp(sigr*EPS_R) - R/steady_state(R);
  RR = R/PAI(+1);
```

To be able to solve the DSGE model, you first need to parse the model file

```
m = nb_dsge('nb_file','frwz_nk.nb');
```

where `frwz_nk.nb` in this case is the name of the model file.

The second thing you need to do is to assign some parameters

```
p.betta = 0.99;
p.kappa = 161;
p.eta   = 10;
p.rhor  = 0.8;
p.sigr  = 0.0025;
p.mu    = 0.03;
p.psi   = 3.1;
m       = assignParameters(m,p);
```

You can get the current assign parameters from the object by using

```
pOut = m.getParameters();
```

When the parameters are assign the next step will be to solve for the steady-state of the model. You need to provide a MATLAB .m file that does solve the steady-state. In our case it must be something like

```
function [ss,param] = frwz_nk_nb_steadystate(~,p)

param = struct();

% Solve steady-state (This is the only part that needs to be changed)
ss.PAI = 1;
ss.Y   = (p.eta - 1)/p.eta;
ss.R   = exp(p.mu)/p.betta*ss.PAI;
ss.RR  = ss.R/ss.PAI;

end
```

See the [Paulsen \(2021\)](#) for more on the steady-state file. Be aware that the endogenous variables of the model with steady-state 0 need not to be assign a value in the steady-state file. Then we can use this file to solve for the steady-state by using

```
m = checkSteadyState(m, 'steady_state_file', 'frwz_nk_nb_steadystate');
```

This method will throw you an error if the .m file does not return the correct steady-state of the model. You can then see the steady-state values of the models by using the [getSteadyState](#) method

```
ss = getSteadyState(m);
```

This method also allows you to get the steady-state value of some user defined expressions of the endogenous variables of the model

```
ss = getSteadyState(m, 'RR*2');
```

If the steady-state is equal to zero for all endogenous variables you do not need to use the [checkSteadyState](#), as the default is to set the steady-state to 0 for all variables.

Then it is time to calculate the derivatives of the model. This can be done using the [derivative](#) method

```
m = derivative(m);
```

The jacobian will be stored in the solution property of the [nb_dsg](#) object at the field jacobian. To get the order of the derivatives use the [getDerivOrder](#) method.

Then to finally solve the model you must use

```
m = solve(m);
```

To print the solution use the [printDecisionRules](#) method. You can also locate the solution of the model on the companion form in the solution property of the [nb_dsg](#) object. Let n be the number of endogenous variables of the model and let the number of innovations (exogenous variables) be given by n^i , then the solution contain the following fields

A

The transition matrix, as a square matrix with size n . The ordering is given by the endo field.

C

The innovation impact matrix with size $n \times n^i$. The ordering is given by the endo and res fields.

endo

Stores the ordering of the endogenous variables in the solution, as a $1 \times n$ cellstr array.

res

Stores the ordering of the innovations (exogenous variables) in the solution, as a $1 \times n^i$ cellstr array.

vcv

The covariance matrix of the innovations. As a identity matrix with size n^i . (To set the standard deviation of some innovation just multiply the innovation in the model file with a parameter.)

ss

A double with size $n \times 1$ storing the steady-state of the model. The ordering is given by the endo field.

obs

The declared observed variables of the model. As a $1 \times O$ cellstr array.

++

Some other fields that is not to useful for the user.

If you do not want to inspect the steady-state or the derivatives of the model, you can call the `solve` method directly without calling `checkSteadyState` and `derivative` first. This will then be done automatically inside the `solve` method instead.

19.2 Optimal monetary policy

When dealing with optimal monetary policy the steps are the same as in the previous section. Here is an example file of a model solved with optimal monetary policy

```
endogenous
DR,PAI,PAI_GAP,PAI_GAP_POL,Y,Y_GAP,R,RR

exogenous
EPS_R

parameters
betta,eta,kappa,sigr,mu,lam_y,lam_dr

model
1 = betta*(1-.5*kappa*(PAI-1)^2)*Y*R(+1)/((1-
.5*kappa*(PAI(+1)-1)^2)*Y(+1)*exp(mu)*PAI(+1));
1-eta+eta*(1-.5*kappa*(PAI-1)^2)*Y+betta*kappa*(1-
.5*kappa*(PAI-1)^2)*(PAI(+1)-1)*PAI(+1)/(1-
.5*kappa*(PAI(+1)-1)^2)-kappa*(PAI-1)*PAI;
```

```

RR = R/PAI(+1);

PAI_GAP      = log(PAI/steady_state(PAI));
PAI_GAP_POL = PAI_GAP + EPS_R;
Y_GAP        = log(Y/steady_state(Y));
DR           = R - R(-1);

planner_objective{discount = 0.99, commitment=1}
.5*(PAI_GAP_POL^2+lam_y*Y_GAP^2+lam_dr*DR^2);

```

It is the planner_objective function that triggers the optimal monetary policy solution, for more on this function see the DSGE->Model file section.

The rest of the steps are unaffected, but now the multipliers are added to the solution so that n in the last section will now be the number of endogenous plus the numbers of multipliers.

19.3 Model file

See corresponding section in [Paulsen \(2021\)](#) or the examples provided in the sections [19.1](#) and [19.2](#).

19.4 Steady-state file

See corresponding section in [Paulsen \(2021\)](#) or the examples provided in the sections [19.1](#) and [19.2](#).

19.5 Calibration

See the sections [19.1](#) and [19.2](#) for how to assign calibrated parameters.

19.6 Estimation

19.6.1 Mode estimation

To do mode estimation you need to first provide some priors.²⁴ It can be given using a struct

```

priors      = struct();
% priors.(paramName) = {start,mean,std,nameOfDist}
priors.theta = {0.5, 0.5, 0.2, 'beta'};
priors.varphi = {0.8, 0.8, 0.1, 'normal'};
priors.phi = {0.5, 0.5, 0.2, 'beta'};
priors.alpha = {0.1, 0, 2}; % Uniform on [0, 2]!
m            = set(m, 'prior', priors);

```

or using the `nb_distribution` class

```

priors(4,1) = nb_distribution();
priors(1,1) = nb_distribution.parameterization(0.5, 0.2^2, 'beta');
set(priors(1,1), 'name', 'theta', 'userData', 0.5);
priors(2,1) = nb_distribution('type', 'normal', 'parameters', {0.8, 0.1}, ...
                             'name', 'varphi', 'userData', 0.8);
priors(3,1) = nb_distribution.parameterization(0.5, 0.2^2, 'beta');
set(priors(3,1), 'name', 'phi', 'userData', 0.5);
priors(4,1) = nb_distribution('type', 'uniform', 'parameters', {0, 2}, ...
                             'name', 'alpha', 'userData', 0.1);
m = set(m, 'prior', priors);

```

²⁴To do Maximum likelihood (ML) estimation specify uniform priors, i.e. do restricted ML estimation.

For more information on how to set the priors see the help on the method `setPrior`. To plot the priors use the `plotPriors` method.

To do the estimation you can use the `estimate` method. To display the estimation results use `print` method. Be aware that you need to call the `solve` method after the estimation to take into account the estimated parameters. A full example follows

```
me = estimate(m);
me.print

% Solve the estimated version (This is not done in
% the estimate method!!)
mes = solve(me);
```

19.6.2 Sampling

To sample from the posterior distribution you can use the `sample` method. The following example illustrate how to use it

```
me = estimate(m);
me.print

% These are the draws made by the sampler!
samplingDraws = 2000;

% This are the number of draws that is kept for IRFs,
% posterior plots, density forecast etc... They are
% randomly selected from the sampled draws from
% chain 1.
usedDraws = 1000;

opt = nb_mcmc.optimset('log',true,'waitbar',true,'thin',2,...
    'draws',samplingDraws,'qFunction','arw','adaptive','recTarget',...
    'accTarget',0.23,'parallel',false,'chains',2);
mp = sample(mes,'draws',usedDraws,'sampler_options',opt);
```

Use the following commands to get more help on the supported sampling options

```
nb_dsge.help('sampler_options')
help nb_mcmc.optimset
```

To check the sampling results you can run some diagnostics, which all uses the full set of sampled draws.

```
% Trace plot and autocorrelation
[~,plotter,pAutocorr] = checkPosterior(mp);
nb_graphSubPlotGUI(plotter);
nb_graphSubPlotGUI(pAutocorr);

% Mean plot
[res,plotter] = meanPlot(mp);
nb_graphSubPlotGUI(plotter);

[res,plotter] = meanPlot(mp,100);
nb_graphSubPlotGUI(plotter);

% Geweke test
res = geweke(mp)

% Gelman Rubin test
[res,plotter] = gelmanRubin(mp);
nb_graphSubPlotGUI(plotter);
```

To plot the posterior against the prior you can use the `plotPosterior` method (which only uses the number of draws selected by the `usedDraws` variable in this example).

```
plotter = plotPosterior(mp,'prior');
nb_graphMultiGUI(plotter);
```

19.6.3 System priors

The system priors implemented are described in the paper Andrle and Benes (2013), "System Priors: Formulating Priors about DSGE Models' Properties. To add system priors, you can use the `setSystemPrior` method or the `systemPrior` option. Remember to first specify the normal priors as described in the section 19.6.1. An example of the first approach is

```
% Parameterize a gamma with mean 0.4 and variance 0.2
[~,param] = nb_distribution.parameterization(0.4,0.2,'invgamma');
yAutoPrior = @(x) log(nb_distribution.invgamma_pdf(x,param{1},param{2}));
ms = setSystemPrior(m,'corr',{'y','y',1,yAutoPrior});
```

while for the second approach you can use

```
ms = set(m,'systemPrior',@yourPriorFunction);
```

where `yourPriorFunction` must be a .m file with the function to call to evaluate your system prior. The function must take two inputs. The `parser` property and the `solution` property (simplified version) of the model object. In this way you can code how to produce any objective from the model and evaluate the prior based on the value of that objective.

In the same way as for the posterior you must sample from the updated priors to get an idea on how your prior and posterior are different. To sample from the updated prior you can use the method `sampleSystemPrior`. Here is an example

```
opt = nb_mcmc.optimset('log',true,'waitbar',true,'thin',2,...
    'draws',2000,'qFunction','arw','adaptive','recTarget',...
    'accTarget',0.23,'parallel',false,'chains',2);
msp = sampleSystemPrior(ms,'sampler_options',opt);
```

To be sure that the sampling has succeeded you need to check that the sampler has converged. This can be done in the same way as for the posterior draws

```
% Trace plot and autocorrelation
[~,plotter,pAutocorr] = checkUpdatedPriors(msp);
nb_graphSubPlotGUI(plotter);
nb_graphSubPlotGUI(pAutocorr);

% Mean plot
[res,plotter] = meanPlot(msp,'updated');
nb_graphSubPlotGUI(plotter);

[res,plotter] = meanPlot(msp,100,'updated');
nb_graphSubPlotGUI(plotter);

% Geweke test
res = geweke(msp,[],'updated');

% Gelman Rubin test
[res,plotter] = gelmanRubin(msp,'updated');
nb_graphSubPlotGUI(plotter);
```

To plot it against the initial prior use

```
plotter = plotUpdatedPriors(msp, 'prior');
nb_graphMultiGUI(plotter);
```

From this you can follow the same steps as for normal estimation, i.e. to do mode estimation and sample from the posterior as described in section 19.6.1 and 19.6.2. To plot the initial prior, the updated prior and the posterior use

```
plotter = plotPosterior(mp, 'updated', 'prior');
nb_graphSubPlotGUI(plotter);
```

19.7 Filtering

To filter the model you must first set the observed variables by using the `set` method

```
m = set(m, 'observables', {'PAI'});
```

Then you need to assign the data to the object, which again can be done by using the `set` method

```
m = set(m, 'data', dataObject);
```

Here the `dataObject` must be a `nb_ts` object at least storing the variables declared as observed.

Then to filter the model you can use the `filter` method

```
m = filter(m);
```

To get more help on the `filter` method write the following in the command window.

```
help nb_dsgc.filter
```

Go to the corresponding section in the DAG documentation for more on the algorithm used.

19.8 DSGE model with break-point

To add a unexpected break-point in the parameter values of the model you can use either the `breakPoint` statement in the model file (see the DSGE->Model file for more on how to do this), or you can use the `breakPoint` method. E.g.

```
m = breakPoint(m, {'kappa'}, 150, '2012Q1');
```

When this method is used you need to call the `solve` method to solve the model. This you need to do even if you have solved the model before calling this method!

19.8.1 Forecast

To forecasting of the model can be done in the same way as in section 21. The state will just follow from the date input you gave to the `breakPoint` method. See the corresponding section in the DAG documentation for more on the algorithm used in the case of a DSGE model with break-points.

19.8.2 IRF

To produce irf you can follow the steps as in section , but you need to use the startingValues and the states input to this method. Default is to use the first state (regime) for both the startingValues and as the state at each period. See the corresponding section in the DAG documentation for more on the algorithm used in the case of a DSGE model with break-points.

19.8.3 Filtering

No changes on how the method is called on the object. See section 19.7. See the corresponding section in the DAG documentation for more on the algorithm used in the case of a DSGE model with break-points.

19.9 Optimal simple rules

This section describes how to find optimal simple rules using the NB toolbox. First we provide you with a simple model which we will use for the examples later.

```
endogenous
y,pie,i,u,e

exogenous
e_u,e_e

parameters
alpha,beta,epsilon,eta,rho_e,rho_u,theta,varphi

model
    y    = y(+1) - eta*(i - pie(+1)) + e;
    pie = beta*pie(+1) + ((1 - alpha/(1 - alpha + alpha*epsilon))
                           *(1 - theta)*(1 - beta*theta)/theta)*(1/eta)
                           + (varphi + alpha)/(1 - alpha)*y + u;
    e   = rho_e*e(-1) + e_e;
    u   = rho_u*u(-1) + e_u;
```

This is the standard 3 equation new keynesian model, but where we for know have left out the interest rate rule. To proceed we need to provide the loss function to minimize. This can be done with the `setLossFunction` method. Remember to also assign all parameters before running the model.

19.9.1 Commitment

If the above model file is stored in the file nk.nb, you can use the following code to find the optimal simple rule under commitment

```
% Parse model
m = nb_dsges('nb_file','nk.nb','silent',false);

% Parameterization
par      = struct();
par.beta = 0.99;
par.eta  = 1;
par.alpha = 0.33;
par.epsilon = 6;
par.theta = 0.67;
par.varphi = 1;
par.rho_u = 0;
```

```

par.rho_e    = 0;
m            = assignParameters(m,par);

% Set loss function
paramL = struct('lambda',0.1);
m      = setLossFunction(m,'0.5*(pie^2 + lambda*y^2)');
m      = assignParameters(m,paramL);

% Optimizer options
opt        = optimset('fmincon');
opt.Display = 'none';

% Initial values of the optimization
init       = struct('gamma_pie',1.2,'gamma_y',0.6, ...
                    'gamma_pie_lag',1.2,'gamma_y_lag',0.6, ...
                    'gamma_e',0,'gamma_u',0);

% Optimal simple rule (commitment)
% This replicate the commitment solution
m_osr_c   = optimalSimpleRules(m, ...
[i = gamma_pie*pie + gamma_y*y + gamma_pie_lag*pie(-1), ...
 ' + gamma_y_lag*y(-1) + gamma_e*e_e + gamma_u*u_e'], ...
'osr_type',      'commitment',...
'init',         init, ...
'lc_discount',  1, ...
'optimset',     opt, ...
'name',         'osr_c');

m_osr_c.print()
m_osr_c.printDecisionRules()

```

Use

```
help nb_dsgc.optimalSimpleRules
```

to get more help on the `optimalSimpleRules` method.

19.9.2 Discretion

Under discretion the rule can only respond to state variables. The first steps are as in the case of the commitment, so given the `nb_dsgc` object `m` we can proceed with

```

% Optimizer options
opt        = optimset('fmincon');
opt.Display = 'none';

% Initial values of the optimization
init       = struct('gamma_pie',1.2,'gamma_y',0.6, ...
                    'gamma_e',0,'gamma_u',0);

% Optimal simple rule (discretion)
% This replicate the discretion solution
m_osr_d   = optimalSimpleRules(m, ...
'i = gamma_pie*pie(-1) + gamma_y*y(-1) + gamma_e*e_e + gamma_u*u_e', ...
'osr_type',      'discretion',...
'init',         init, ...
'lc_discount',  1, ...
'optimset',     opt, ...
'name',         'osr_d');

m_osr_d.print()
m_osr_d.printDecisionRules()

```

19.9.3 Calculate loss

To calculate the loss of any solved `nb_dsge` object m you can use the `calculateLoss` method. If your model is not going to be solved under optimal monetary policy or optimal simple rules, it is important to set a third input to the `setLossFunction` method to false (which must of course be called before `calculateLoss`).

20 Analysis

In this section we use the following example.²⁵

```
% Declare the variables
variables = {'Var1', 'Var1', 'Var3'};

% Formulate model
options          = nb_var.template();
options.data     = data;
options.dependent = variables;
options.nLags    = 2;
options.constant = false;

% Create model object
model = nb_var(options);

% Estimate model
model = estimate(model);

% Identify model
model = set_identification(model, 'cholesky', ...
    'ordering', variables);
model = solve(model);
```

20.1 Impulse response functions

To produce impulse responses, use the `irf` method.

```
[irfs, irfsBand, plotter] = irf(model, 'perc', 0.68, 'replic', 1000);
plotter.graphInfoStruct();
```

20.2 Shock decomposition

To perform shock decomposition, use the `shock_decomposition` method:

```
[decomp, decompBand, decompPlotter] = shock_decomposition(model, ...
    'variables', variables, ...
    'startDate', '', ...
    'endDate',   '', ...
    'Perc',      0.9);
decompPlotter.graph();
```

20.3 Forecast error variance decomposition (FEVD)

To perform FEVD, use the `variance_decomposition` method:

²⁵The data object is the same as that of section 13.

```
% Variance decomposition
[decomp, decompBand, decompPlotter, decompPlotterPerc] = ...
    variance_decomposition(model, 'variables', variables, ...
        'perc', 0.9,
        'output', 'actual');
decompPlotter.graph();

fields = fieldnames(decompPlotterPerc);
for i = 1:length(fields)
    graphSubPlots(decompPlotterPerc.(fields{i}));
end
```

20.4 Model Moments

Model objects have methods for calculating empirical, theoretical and simulated moments, i.e. mean, (auto)covariance and (auto)correlation of the model data.

20.4.1 Empirical moments

To calculate empirical moments of a model, use the [empiricalMoments](#) method:

```
[mean, cov, autoCov1, autoCov2] = empiricalMoments(...  
    model, 'type', 'covariance'); % or 'correlation'
```

The first output is the mean, the second the correlation/-covariance matrix and the following outputs are autocorrelation/-covariance matrices at increasing time lags. The number of output variables you supply decides how many such autocorrelation/-covariance matrices are calculated.

For more information about the available options, see the [empiricalMoments](#) documentation.

20.4.2 Simulated moments

To calculate simulated moments of a model, use the [simulatedMoments](#) method:

```
[mean, cov, autoCov1, autoCov2] = simulatedMoments(...  
    model, ...  
    'type', 'covariance', ...  
    'nSteps', 100, ...  
    'pDraws', 1, ...  
    'draws', 100);
```

The first output is the mean, the second is the correlation/-covariance matrix and the following outputs are autocorrelation/-covariance matrices at increasing time lags. The number of output variables you supply decides how many such autocorrelation/-covariance matrices are calculated.

For more information about the available options, see the [simulatedMoments](#) documentation.

20.4.3 Theoretical moments

To calculate theoretical moments of a model, use the [theoreticalMoments](#) method:

```
[mean, corr, autoCorr1, autoCorr2] = theoreticalMoments(...  
    model, ...  
    'type', 'correlation', ...  
    'maxIter', 1000, ...  
    'pDraws', 1, ...
```

```
'perc', [0.1, 0.3, 0.5, 0.7, 0.9]);
```

The first output is the mean, the second the correlation/-covariance matrix and the following outputs are autocorrelation/-covariance matrices at increasing time lags. The number of output variables you supply decides how many such autocorrelation/-covariance matrices are calculated.

For more information about the available options, see the [theoreticalMoments](#) documentation.

21 Forecasting

21.1 Unconditional forecast

When doing unconditional forecast we can skip the identification part, even for VARs. Instead we just solve it without identification:²⁶

```
model = solve(model);
```

After having the solution of the model we can produce forecast using the [forecast](#) method;

```
model = forecast(model,8);
```

Where the second input is the forecast horizon. To plot the forecast we can use the [plotForecast](#) method:

```
plotter = plotForecast(model);
plotter.graphSubPlots
```

To produce density forecast without parameter uncertainty the following code can be used:

```
model = forecast(model,8,'draws',1000);
```

Here 1000 draws from the residuals/shocks are used to produce density forecast. The forecast can be plotted with the same code as before. If we also want to take parameter uncertainty into account it can be done by:

```
model = forecast(model,8,'draws',100, ...
    'parameterDraws',1000, ...
    'method','bootstrap');
```

Where the 'parameterDraws' are the number of draws from the distribution of the parameters. Be aware that there will be done 100 residual draws per parameter draws in this example. The 'method' option selects the method to draw the parameters from its unknown distribution. For models estimated with classical methods, a bootstrapping method must be used, while models estimated with bayesian methods will sample directly from the posterior distribution.

To do in-sample recursive forecast you need to provided the 'fcstEval' input, here set to calculate squared errors:

```
model = forecast(model,8,'fcstEval','SE');
```

²⁶If the model doesn't need to be identified this amounts to the same.

To plot the recursive forecast the 'hairyplot' option may be used:

```
plotter = plotForecast(model,'hairyplot');  
plotter.graphSubPlots
```

If on the other hand the 'recursive_estim' estimation input is set to 1 when initializing the model, you can produce out-of-sample recursive forecast with the same code as for in-sample recursive forecast.

It is also possible to do recursive density forecast by combining the above options.

21.2 Conditional forecast

See the example script at \\NBTOOLBOX\\Examples\\Econometrics\\VAR\\nb_var_ols_condForecast.m

21.3 Forecast evaluation

See section 21.1

21.3.1 Mincer-Zarnowitz test

To do the Mincer-Zarnowitz test we can use the following code, remember that recursive forecast must have been done

```
[test,pval,printed] = mincerZarnowitzTest(model)
```

21.3.2 Diebold-Mariano test

Let's say we have two models forecasting the same variable. Both being stored as nb_model_generic objects, i.e. being of any type of model which is part of the NBToolbox. Call them model1 and model2. Both must have produced recursive forecast for an overlapping period of time. Then we can run the Diebold-Mariano test for differences in forecast performance as follows:

```
[test,pval,res] = dieboldMarianoTest(model2,model)
```

21.3.3 PIT

If recursive density forecast has been produced by a model we can evaluate it using a PIT graph as follows:

```
[pit,plotter] = getPIT(model);%, '1994Q1', false  
plotter.graph()
```

Where the bars should level up with the plotted line for the density forecast to be performing well.

21.4 Forecast combination

Now we are interested in combining forecast from different models. So first we initialize 4 VARs with different lag lengths:

```
% Formulate model  
options = nb_var.template(4);
```

```

options.data          = data;
options.dependent     = {'Var1', 'Var2', 'Var3'};
options.nLags         = {1,2,3,4};
options.recursive_estim = 1;

% Create model object
models = nb_model_generic.initialize('nb_var',t);

% Estimate and solve model
model = estimate(model);
model = solve(model);

```

Then we want to produce recursive point forecast:

```
models = forecast(models,8,'fcstEval','SE');
```

Before we do the forecast combination we need to create a `nb_model_group` object:

```
modelGroup = nb_model_group(models);
```

At last we use the `combineForecast` method to do the forecast combination:

```
[modelGroup,weights,plotter] = combineForecast(...  
    modelGroup,...  
    'allPeriods', 1,...  
    'fcstEval',   {'SE'},...  
    'type',       'MSE');
```

Here we indicate by the 'allPeriods' input that we want to construct recursive forecast for the combined forecast as well, and not only for the last period. Be aware that if a model does not forecast a series no error will be provided, the model will only be given zero weight. The same goes for a model that hasn't produced forecast at a given period in the recursive sample shared across models.

The 'type' option selects the method to construct the weights. In this example we use mean squared error(MSE).

Using the output we can create a plot of the weights at different horizons, in this example at horizon 1:

```
graph(plotter(1));
```

We can plot the final periods combined forecast with the following code:

```
plotter = plotForecast(modelGroup);  
plotter.graphSubPlots();
```

To compare it to the individual forecast we can plot all of them in the same graph:

```
plotter = plotForecast(models);  
plotter.graphSubPlots();
```

It is also possible to combine density forecast with the use of the 'density' and 'draws' options, but then all the models must have produced density forecast.

21.5 Aggregate forecast

Let's say we want construct an aggregate forecast. I.e. if we have produced forecast on subcomponents of a series and we want to aggregate it to produce a forecast on

the aggregated series. This can be done with the `aggregateForecast` method of the `nb_model_group` class.

First we need some data:

```
% Simulate some artificial data
nDis    = 4;
obs     = 100;
lambda  = [ 0.5,  0.1,  0.3,  0.6,  0.2,  0.2, -0.1, 0;
            0.5, -0.1, -0.2,  0.4,   0,  0.1, -0.2, 0;
            0.6, -0.2,  0.1, -0.2,   0,  0.4, -0.1, 0;
            -0.2, -0.4,   0,  0.7,   0,      0,      0, 0];
rho     = [1;1;1;1];
sim     = nb_ts.simulate('2012M1',obs,{ 'VAR1', 'VAR2', ...
    'VAR3', 'VAR4'},1,lambda,rho);
weights = [0.3,0.2,0.4,0.1];
agg     = sum(bsxfun(@times,weights,double(sim)),2);

% Transform to nb_ts object
data   = nb_ts([double(sim),agg],'', '2000Q1',...
    {'Sim1','Sim2','Sim3','Sim4','Agg'},false);;
```

The data consist of subcomponents of consumption of goods. We want to produce forecast of each subcomponent using an AR(1,0,0) model. So first we set up all the models:

```
% Set options shared by all models
options                     = nb_arima.template;
options.AR                  = 1;
options.MA                  = 0;
options.integration         = 0;
options.constant             = 1;
options.recursive_estim     = 1;
options.recursive_estim_start_date = '2010Q1';
options.data                = data;
models                      = nb_arima(options);

% Replicate the model
models = repmat(models,1,4);
nModels = numel(models);

% Loop through for each variable
variables = data.variables(1:4);
for ii = 1:nModels
    models(ii) = set(models(ii), 'dependent', variables(ii));
end

% Estimate and solve all models
models = estimate(models);
models = solve(models);
```

Secondly we want to produce recursive forecast:

```
models = forecast(models,8,'fcstEval',{'SE'});
```

Then using the known share of each subcomponent we aggregate the forecast:

```
% Construct a nb_model_group object
modelGroup = nb_model_group(models);

% Do the aggregation
modelGroup = aggregateForecast(modelGroup, ...
    'weights',      weights, ...
    'newVar',       'Agg', ...
```

```

    'variables',    variables, ...
    'fcstEval',    {'SE'});

```

The 'variables' input must be a cellstr with the same size as the number of models to aggregate. I.e. it selects which variable to pick from each model to be aggregated. In this example this is trivial, but the forecast may come from models with more than just one variable. The 'newVar' option is the name of the aggregated variable. As you probably have the history of this variable it will be smart to store this in the data property of the different models with the same name. This is to be sure that the forecast evaluation compares to the exact observations of the aggregated variable.

Then we can plot the aggregated forecast and the forecast of each subcomponent as follows:

```

% Plot the aggregated forecast with history
plotter = plotForecast(modelGroup);
nb_graphSubPlotGUI(plotter);

% See the individual forecast
plotter = plotForecast(models);
nb_graphSubPlotGUI(plotter);

```

It is also possible to aggregate density forecast. This is done by:

```

% Density Forecast (recursive)
models = forecast(models,8, ...
    'fcstEval',{'SE'},...
    'draws', 1000, ...
    'perc', []);

% Aggregate models
modelGroup = nb_model_group(models);
modelGroup = aggregateForecast(modelGroup, ...
    'density', true, ...
    'method', 'copula',...
    'weights', weights, ...
    'newVar', 'Agg',...
    'variables', variables, ...
    'fcstEval', {'SE'}, ...
    'nLags', 4, ...
    'perc', []);

% Plot the aggregated forecast with history
plotter = plotForecast(modelGroup);
nb_graphSubPlotGUI(plotter);

% See the individual forecast
for ii = 1:length(models)
    plotter = plotForecast(models(ii));
    nb_graphSubPlotGUI(plotter);
end

```

The important options in this case being 'density', 'method' and 'nLags'. The 'density' option must be set to true, if not only the mean forecast is used to produce an aggregated point forecast. The 'method' option can be used to set the algorithm to make draws from the densities of the subcomponents. The 'copula' option estimate the auto-correlation for the number of needed periods to make draws from the different densities that are consistent with historically observed relation between the subcomponents (cross correlation and cross autocorrelation) and individual autocorrelation. The other option for 'method' is to assume they are perfectly correlated, which may overestimate the variance in the aggregated forecast. When 'method' is set to 'copula' the 'nLags' input sets the number of periods of history to condition on when drawing from the densities of

the subcomponents. A high number here is needed if the subcomponents display high autocorrelation.

21.6 Data transformation and reporting

For point forecast variables may be transformed with any type of expression after the forecast has been produced, this may no longer be true for density forecast. For example when calculating percentiles (for non-linear expressions).

If the forecast should be evaluated based on some transformation of variables it is also important to do them in code, at least it is much easier than to write your own code for it! (Both point and density.)

Here is an example:

```
% Generate some artificial data
obs      = 100;
lambda   = [0.5, 0.1, 0.3, 0.2, 0.2, -0.1;
            0.5, -0.1, -0.2, 0, 0.1, -0.2;
            0.6, -0.2, 0.1, 0, 0.4, -0.1];
rho      = [1; 1; 1];
vars     = {'VAR1', 'VAR2', 'VAR3'};
sim      = nb_ts.simulate('1990Q1', obs, vars, 1, lambda, rho);
data     = igrowth(sim/100+0.01, ones(1, 3)*100, 1);

% VAR Options
t         = nb_var.template();
t.data    = data;
t.constant = 0;
t.nLags   = 2;

% Create model and estimate
model = nb_var(t);

% Do transformations (To store shift/trend)
expressions = {%
    Name, expression, shift/trend, description
'VAR1_growth', 'growth(VAR1)', {'avg'}, ''
'VAR2_growth', 'growth(VAR2)', {'avg'}, ''
'VAR3_growth', 'growth(VAR3)', {'avg'}, ''
};
fcstHor = 8; % The forecast horizon
[model, plotter] = model.createVariables(expressions, ...
                                         fcstHor);
nb_graphInfoStructGUI(plotter);

% Assign inverse transformations
% (shift/trend will be added automatically)
expressions = {%
    Name, expression, description
'VAR1_L', 'igrowth(VAR1_growth, VAR1)', ''
'VAR2_L', 'igrowth(VAR2_growth, VAR2)', ''
'VAR3_L', 'igrowth(VAR3_growth, VAR3)', ''
'VAR1', 'VAR1_L', ''
'VAR2', 'VAR2_L', ''
'VAR3', 'VAR3_L', ''
};
model = set(model, 'reporting', expressions);

% Check reporting
model = checkReporting(model);

% Formulate model
VARvars = {'VAR1_growth', 'VAR2_growth', 'VAR3_growth'};
model   = set(model, 'dependent', VARvars);

% Estimate
model = estimate(model, ...
                  'recursive_estim', true, ...)
```

```

'recursive_estim_start_date','2009Q4');

print(model)

% Solve model and do point forecast
model = solve(model);
model = forecast(model,8, ...
    'fcstEval','SE',...
    'varOfInterest',{'VAR1','VAR2','VAR3'});
plotter = plotForecast(model,'hairyplot');
plotter.set('startGraph','2005Q1');
nb_graphSubPlotGUI(plotter);

```

21.7 Evaluate forecast from model

To evaluate the forecast from the model at specific dates you can use the `evalFcstAtDates` method of the class `nb_model_forecast` (the superclass of all models producing forecast). Here is a full example code for doing this:

```

% Generate some data
draws = randn(100,1);

% ARIMA(1,0,0)
sim1 = filter(1,[1,-0.5],draws);
data = nb_ts(sim1,[],'1995Q1',{'Sim1'});

% Set up model
t = nb_arima.template();
t.data = data;
t.dependent = data.variables;
t.integration = 0;
t.AR = 1;
t.MA = 0;
t.recursive_estim = 1;
t.recursive_estim_start_date = '2001Q1';

% Estimate and forecast model
m = nb_arima(t);
m = estimate(m);
m = solve(m);
m = forecast(m,5,'fcstEval','SE');

% Set the dates for where to evaluate the model
dates = {
    '20170305'
    '20170605'
    '20170905'
    '20171205'
    '20180305'
    '20180605'
    '20180905'
    '20181205'
};

% Get scores from the model
[Mdata,errorD] = evalFcstAtDates(m, ...
    dates,{ 'rmse'},false, false);
if ~isempty(errorD)
    disp(toString(errorD))
end

```

The output is an object of class `nb_data` with the score of the wanted forecast evaluation. For more on this method type `help nb_model_forecast.evalFcstAtDates` in the command window.

22 Model selection

To select the M best VAR/ARIMA models out of a suit of VAR/ARIMA models, use the `nb_model_selection_group` class. To initialize a `nb_model_selection_group` object, start by calling the `nb_model_selection_group()` method. Set class property to 'var' for VAR models and 'arima' for ARIMA models.

```
% Generate some data
obs      = 100;
lambda   = [0.5, 0.1, 0.3, 0.2, 0.2, -0.1;
            0.5, -0.1, -0.2, 0, 0.1, -0.2;
            0.6, -0.2, 0.1, 0, 0.4, -0.1];
rho      = [0.5; 0.5; 0.5];
sim      = nb_ts.simulate('1990Q1', obs, {'VAR1', 'VAR2', ...
                                         'VAR3'}, 1, lambda, rho);
data     = igrowth(sim/100 + 0.01, 100);

% Formulate the model
modelGroup = nb_model_selection_group();
modelGroup = set(modelGroup, ...
                 'data', ...
                 'varOfInterest', ...
                 'modelVarOfInterest', ...
                 'variables', ...
                 { ...
                  'G_VAR1', ...
                  'G_VAR1_GAP', ...
                  {'G_VAR2_GAP', ...
                   'G_VAR3_GAP'}});
```

If you want the forecast to be evaluated on some transformation of the variable, assign the desired transformations and upload the relevant variables to the model group:

```
% Assign desired transformations (shift/trend will be stored)
transformations = {%
    Name, expression, shift/trend, description
    'G_VAR1_GAP', 'pcn(VAR1)', {'avg'}, '% growth gap'
    'G_VAR2_GAP', 'pcn(VAR2)', {'avg'}, '% growth gap'
    'G_VAR3_GAP', 'pcn(VAR3)', {'avg'}, '% growth gap'
};

% Upload data to the model group
modelGroup = createVariables(modelGroup, transformations, 8);
```

If you wish to add the shift/trend from the transformations above when producing forecasts, you can assign the inverse transformation by reporting the variables. This will automatically add the shift/trend to the results:

```
% Report variables
reporting = {%
    Name, expression, description
    'G_VAR1', 'G_VAR1_GAP', '% growth'
};

% Add the reporting to the model group
modelGroup = set(modelGroup, 'reporting', reporting);
modelGroup = checkReporting(modelGroup);
```

Run the model selection and plot the number of times each variable has been chosen for the M best VAR/AR models:

```
% Initialize model object to do model selection
[modelGroups, p] = modelSelection(modelGroup);

% Plot histogram over most selected variables
p.graph
```

23 More examples

More examples can be found in NBTOOLBOX\Examples\Econometrics.

24 Code references

We thank a lot of contributors for making their code open source, as without them we could not have made this toolbox. Here we have to our best effort tried to list the most important once:

Altman, Y and Woodford, Oliver Export_fig. See the [export_fig](#) function.²⁷

Altman, Y Find java objects contained within a specified java container or Matlab GUI handle. See the [findjobj](#) function.

Benes, Jaromir Do x12 Census adjustment to time series using x12awin.exe. See the [x12.x12](#) function, which is a modified version of the corresponding function which is part of the [IRIS toolbox](#). x12awin.exe is a program developed by the U.S. Department of Commerce, U. S. Census Bureau.

Binning, Andrew Identification of VAR model using combination of zero and sign restrictions. Available [here](#). See [nb_var.ABidentification](#).

Chen, S. Samuel Particle Swarm Optimization. Please see the [nb_pso](#) and [pso.do](#) function. Available [here](#).

Dynare NB toolbox makes it possible to use the [nb_dsge](#) class as a overhead of the the [Dynare toolbox](#), so as to make it possible to compare models written in different languages. Parsing and solving is then done with Dynare, but the other analysis, as IRFs, shock decomposition etc. is using code from the NB toolbox. You need to download and add Dynare to the MALTAB run path yourself to able to run Dynare.

Fink, Martin and Ahn, SeHyoun Automatic differentiation of 1st. See the [myAD](#) class.

Gleich, David F. Parts of the Graph Algorithms package GAIMC. Please use [this](#) link to get the original code.

Hoffman, Matthew D. The No-U-Turn-Sampler implementation in MATLAB. See [nb_mcmc.NUTS](#) and [nb_mcmc.dualAveraging](#).

²⁷NB toolbox provide additional functionality. You may flip the PDF with the '-noflip' option, and you may create a portrait PDF using the [portraitPDF](#) function.

References

Paulsen, K. S. (2021) *NB toolbox: Theory, algorithms and implementations.*

25 Library

◆ nb_startup

```
nb_startup(type,silent)
```

Description:

Start up function of the NB toolbox.

This function utilizes a simplified version of the cprintf function made by Yair M. Altman.

Input:

- type : With this input you can specify which functionalities you want
 - 'all' : Add all the functionalities.
 - 'dg' : Only add the data managment and graphics functions and classes.
 - 'full' : Will add all the functionalities of the toolbox except the GUI part.
 - 'g' : Only add the basic plot functions and classes.
 - 'gui' : Add all the functionalities. Same as 'all'.
- silent : If you give 'silent' it will not print anything on the screen.

Output:

- All the wanted function and classes added to the MATLAB path.

Examples:

```
nb_startup  
nb_startup('fg')  
nb_startup('full','silent')
```

Written by Kenneth S. Paulsen

25.1 Date classes and functions

- nb_date
- nb_dateInExpr
- nb_dateminus
- nb_dateplus
- nb_day
- nb_easter
- nb_isDate
- nb_month
- nb_quarter
- nb_sameDateFormat
- nb_semiAnnual
- nb_week
- nb_xlsDates2FAMEVintage
- nb_year

■ nb_date

Go to: [Properties](#) | [Methods](#)

The superclass of all the different dates at different frequencies

This class also includes some static functions which are general for all date frequencies.

Subclasses:

`nb_day, nb_week, nb_month, nb_quarter, nb_semiAnnual nb_year`

Constructor:

`obj = nb_date %` Constructs an empty date object

Input:

No inputs

Output:

No outputs

Examples:

`obj = nb_date;`

See also:

`nb_year, nb_semiAnnual, nb_quarter, nb_month, nb_week, nb_day`

Written by Kenneth S. Paulsen

Properties:

- `baseYear`
- `dayOfWeek`
- `leapYear`

- **baseYear** ↑

The base year. 2000.

- **dayOfWeek** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

- **leapYear** ↑

Indicator for leap year. 1 if leap year.

Methods:

- ascensionDay
- colon
- easter
- fromxlsDates2freq
- getFrequencyAsInteger
- getNr
- intersect
- isFirstPeriod
- isempty
- min
- sort
- toString
- unique
- cell2Date
- date2freq
- format2string
- getEarliestDate
- getFrequencyAsString
- holidays
- isDate
- isLeapYear
- ismember
- pentecost
- toDate
- today
- vintage2Date
- christmas
- disp
- freqPlus
- getFreq
- getLatestDate
- initialize
- isEqualFreq
- isPeriod
- max
- setDayOfWeek
- toFormat
- union

► ascensionDay ↑

```
day = ascensionDay(obj,~)
```

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_date.easter](#), [nb_date.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

► cell2Date ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.
- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'
      '2011Q3'
      '2011Q4'
      '2012Q1'
      '2012Q2'
      '2012Q3'
      '2012Q4'
      '2013Q1'
      '2013Q2'}
```

```
a = nb_date.cell2Date(g,4)
```

```
a =
```

```
'2011Q2'
'2011Q3'
'2011Q4'
'2012Q1'
'2012Q2'
'2012Q3'
'2012Q4'
'2013Q1'
'2013Q2'
```

Written by Tobias Ingebrigtsen

► christmas ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_date.ascensionDay](#), [nb_date.pentecost](#), [nb_date.easter](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **colon** ↑

cellStr = colon(a,d,b)

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;  
dates = a:2:b;
```

Written by Kenneth S. Paulsen

► **date2freq** ↑

[startDate,frequency,type] = nb_date.date2freq(dates,xls)

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'  
semiannually : 'yyyySs' or 'dd.mm.yyyy'  
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'  
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'  
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'  
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
> otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- type : 1 if the dates input is on the xls format, otherwise 0

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

► **disp** ↑

`disp(obj)`

Description:

Sets how the `nb_date` object is displayed, and all subclasses of the `nb_date` class

Input:

- `obj` : An object of class `nb_date` or of a subclass of the `nb_date` class.

Written by Kenneth S. Paulsen

► **easter** ↑

`day = easter(obj,type)`

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- `obj` : An object of class `nb_date`.
- `type` : 'all' or 'sunday' (default).

Output:

- `days` : All the holidays as a vector of `nb_day` objects.

See also:

[nb_easter](#), [nb_date.ascensionDay](#), [nb_date.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj,format)
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- d : Day of the date. Examples:

```
> 'dd'   : '01', '10'
> 'd'    : '1', '0'
> 'd(d)': '1', '10'
- w : Week of date. Same options as for d.
- m : Month of date. Same options as for d.
```

- q : Quarter of date. Examples:

```
> 'q'   : '1', '4'
- h : Half year of date. Same options as for h.
- y : Year of date.
      > 'yyyy' : '2016'
      > 'yy'   : '16'
      < 'y'    : '6'
```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'
 > 'english' : 'Q'
- 'Weektext' : > 'norwegian' : 'Uke'
 > 'english' : 'Week'
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d      = nb_day.today();
date = nb_date.format2string(d,'dd.mm.yyyy');
date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m      = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q      = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y      = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');
```

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

► **freqPlus ↑**

```
obj = freqPlus(obj,incr,freq)
```

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

► **getFreq** ↑

```
freq = nb_date.getFreq(stringDate)
```

Description:

Get the frequency of a string date, given that it is on the nb_ts date format.

Input:

- stringDate :

Supported date formats:

```
> yearly      : 'yyyy'  
> semiannually : 'yyyySs'  
> quarterly   : 'yyyyQq'  
> monthly     : 'yyyyMm(m)'  
> weekly      : 'yyyyWw(w)'  
> daily       : 'yyyyMmDd(d)'
```

Output:

```
- freq      :  
  > yearly    : 1  
  > semiannually : 2  
  > quarterly   : 4  
  > monthly     : 12  
  > weekly      : 52  
  > daily       : 365
```

Examples:

```
freq = nb_date.getFreq('2012'); % Will return 1
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger** ↑

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Output:

- frequency : The frequency as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

► **getLatestDate** ↑

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

► **getNr** ↑

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_date.easter](#), [nb_date.ascensionDay](#), [nb_date.pentecost](#)
[nb_date.christmas](#)

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

► **intersect** ↑

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

► **isDate** ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth Sæterhagen Paulsen

► **isEqualFreq** ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

► **isFirstPeriod** ↑

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)

m = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sætherhagen Paulsen

► **isLeapYear** ↑

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)
```

Written by Kenneth Sæterhagen Paulsen

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty, which is always true.

Input:

- obj : An object of class nb_date

Output:

- ret : 1 if empty, 0 else

Examples:

```
ret = objisempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

► **max** ↑

```
obj = nb_date.max(obj1,obj2)
```

Description:

Find the latest date of the two. The frequency must be the same!

Input:

- obj1 : A scalar nb_date object.
- obj2 : A scalar nb_date object.

Output:

- obj : The last of the two dates, as a nb_date object.

Written by Kenneth Sæterhagen Paulsen

► **min** ↑

```
obj = nb_date.min(obj1,obj2)
```

Description:

Find the first date of the two. The frequency must be the same!

Input:

- obj1 : A scalar nb_date object.
- obj2 : A scalar nb_date object.

Output:

- obj : The first of the two dates, as a nb_date object.

Written by Kenneth Sæterhagen Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_date.easter](#), [nb_date.ascensionDay](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **setDayOfWeek** ↑

`obj = setDayOfWeek(obj,dayOfWeek)`

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_date.dayOfWeek](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **sort** ↑

`[obj,ind] = sort(obj)`

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.
- ind : The index of the sorting.

Written by Kenneth Sæterhagen Paulsen

► **toDate** ↑

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily	:	'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'
> Weekly	:	'dd.mm.yyyy' and 'yyyyWw(w)'
> Monthly	:	'dd.mm.yyyy' and 'yyyyMm(m)'
> Quarterly	:	'dd.mm.yyyy' and 'yyyyQq'
> Semiannually	:	'dd.mm.yyyy' and 'yyyySs'
> Yearly	:	'dd.mm.yyyy' and 'yyyy'
> Secondly	:	'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss' (only supported if this version of the toolbox includes the nb_second class)

- frequency :

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2

```
> Yearly      : 1  
> Secondly    : 31536000
```

Output:

```
- dObj      :  
  > Daily     : An nb_day object  
  > Weekly    : An nb_week object  
  > Monthly   : An nb_month object  
  > Quarterly : An nb_quarter object  
  > Semiannually : An nb_semiAnnual object  
  > Yearly    : An nb_year object  
  > Secondly   : An nb_second object
```

Examples:

```
dObj = nb_date.toDate('2012M1D1', 365);  
dObj = nb_date.toDate('2012M1', 12);  
dObj = nb_date.toDate('2012Q1', 4);  
dObj = nb_date.toDate('2012S1', 2);  
dObj = nb_date.toDate('2012', 1);
```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start, finish, format, language, first)
```

Description:

Convert dates vector to a given format.

Input:

```
- start      : An object of a subclass of the nb_date class.  
- finish     : An object of a subclass of the nb_date class.  
- format     : See the format input to the nb_date.format2string  
               method.  
- language   : 'norwegian' or 'english' (default). Only important for the  
               'monthtext' or 'Monthtext' patterns.  
- first      : Give false to return the latest period when converted  
               to a higher frequency. Default is true, i.e. first  
               period.
```

Output:

- vint : A cellstr.

See also:

[nb_date.format2string](#)

Written by Kenneth Sæterhagen Paulsen

► **toString** ↑

date = toString(obj)

Description:

Transform the nb_date object to a string. Will always return 'empty date object'

Input:

- obj : An object of class nb_date

Output:

- string : 'empty date object'

Examples:

Written by Kenneth S. Paulsen

► **today** ↑

obj = nb_date.today(freq)

Description:

Get the current day as a nb_day object.

Input:

- freq : The wanted frequency of the date of today.

1, 2, 4, 12, 52, {365}

Output:

obj : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

► **union** ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth Sæterhagen Paulsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |  
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

■ nb_dateInExpr

Go to: [Properties](#) | [Methods](#)

A class representing a date that can be used in nb_eval to get the end dates of time-series used in an expression.

Constructor:

```
obj = nb_dateInExpr(date)
```

Input:

- date : An object of class nb_date or a one line char that can be converted to a nb_date object using nb_date.date2freq.

Output:

- obj : An object of class nb_dateInExpr.

Examples:

```
obj = nb_dateInExpr('2012Q1');
```

See also:

[nb_date](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [date](#)

- **date** ↑

The date the object represents.

Methods:

• abs	• acos	• acosd	• acosh	• acot
• acotd	• acoth	• acsc	• acscd	•acsch
• and	• append	• asec	• asecd	• asech
• asin	• asind	• asinh	• atan	• atand
• atanh	• avgOAF	• bkfilter	• bkfilter1s	• callfun
• ceil	• conj	• convert	• corr	• cos
• cosd	• cosh	• cot	• cotd	• coth
• cov	• csc	• cscd	• csch	• cumprod
• cumsum	• demean	• demeanMoving	• denton	• deptcat
• detrend	• diff	• disp	• egrowth	• epcn
• eq	• exp	• expand	• expm1	• extrapolate
• fillNaN	• floor	• ge	• growth	• gt
• h2l	• horzcat	• horzcatfast	• hpfilter	• hpfilter1s
• iegrowth	• iegrowthnan	• iepcn	• iepcnnan	• igrowth
• igrowthnan	• interpolate	• ipcn	• ipcnnan	• isfinite
• isnan	• kurtosis	• lag	• ldivide	• le
• lead	• log	• log10	• log1p	• log2
• lt	• mavg	• max	• mean	• median
• merge2Series	• min	• minus	• mldivide	• mpower
• mrdivide	• mstd	• mtimes	• ne	• not
• or	• pca	• pca_func	• pcn	• plus
• pow2	• power	• q2y	• rdivide	• reIndex
• real	• reallog	• realpow	• realsqrt	• ret
• rlag	• round	• sec	• secd	• sech
• set2Value	• setNan2Zero	• sgrowth	• sin	• sind
• sinh	• skewness	• sqrt	• std	• stdise
• subAvg	• subSum	• sum	• sumOAF	• tan
• tand	• tanh	• times	• uminus	• undiff
• uplus	• var	• vertcat	• window	• x12
• x12Census				

► **abs** ↑

```
obj = abs(obj)
```

Description:

Absolute value

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acos** ↑

```
obj = acos(obj)
```

Description:

Take inverse cosine, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acosd** ↑

```
obj = acosd(obj)
```

Description:

acosd(obj) is the inverse cosine, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acosd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acosh** ↑

```
obj = acosh(obj)
```

Description:

acosh(obj) is the inverse hyperbolic cosine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acosh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acot** ↑

```
obj = acot(obj)
```

Description:

Take inverse cotangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

acotd(obj) is the inverse cotangent, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acotd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acoth** ↑

```
obj = acoth(obj)
```

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acoth(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsc** ↑

```
obj = acsc(obj)
```

Description:

Take acsc

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acsc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsqd** ↑

```
obj = acscd(obj)
```

Description:

acscd(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acscd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsch** ↑

```
obj = acsch(obj)
```

Description:

acsch(obj) is the inverse hyperbolic cosecant

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acsch(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **and** ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr

Output:

- a : An object of class nb_objectInExpr.

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **append ↑**

```
obj = append(obj,aObj)
obj = append(obj,aObj,priority)
```

Description:

Append aObj to obj with a given priority.

Input:

- obj : A nb_objectInExpr object.
- aObj : A nb_objectInExpr object.
- priority : 'first' or 'second'. If 'first' all the observations from obj is used before the observations from aObj, otherwise it is reverse. 'first' is default.

Output:

- obj : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asec** ↑

```
obj = asec(obj)
```

Description:

Inverse secant, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asecd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asech(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sine, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asind(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asinh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the tangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = atand(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent

Input:

```
- obj : An object of class nb_objectInExpr
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Examples:

```
out = atanh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **avgOAF** ↑

```
obj = avgOAF(obj,avgFreq)
```

Description:

Take the average over a lower frequency.

Input:

```
- obj : An object of class nb_objectInExpr  
- avgFreq : Any
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **bkfilter** ↑

```
obj = bkfilter(obj,low,high)
```

Description:

Do band pass filtering

Input:

- obj : An object of class nb_objectInExpr
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **bkfilterls** ↑

```
obj = bkfilterls(obj,low,high)
```

Description:

Do one sided band pass-filtering.

Input:

- obj : An object of class nb_objectInExpr
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **callfun** ↑

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the object.

Input:

- obj : An object of class nb_dateInExpr.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- 'frequency' : If the operator should act on the data of the object, as it was on another frequency, you set this option to that frequency. Either 1 (yearly), 2 (semi-annually), 4 (quarterly), 12 (monthly), 52 (weekly) or 365 (daily).

Caution: It will be applied to the obj input as well as all optional inputs being a nb_dateInExpr object.
- 'interpolateDate' : See the nb_dateInExpr.convert method for more on this option. Default is 'start'.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- varargout : The output will be given as return by the func function when applied to the input objects.

Examples:

```
d1 = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2 = callfun(d,d,'func','plus') % Same as d2 = d + d !
```

Written by Kenneth Sæterhagen Paulsen

► ceil ↑

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = ceil(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **conj** ↑

obj = conj(obj)

Description:

conj(obj) is the complex conjugate

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = conj(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **convert** ↑

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_dateInExpr object

Input:

```
- obj      : An object of class nb_dateInExpr  
- freq    : The new frequency of the data. As an integer:  
             > 1    : yearly  
             > 2    : semi annually  
             > 4    : quarterly  
             > 12   : monthly  
             > 52   : weekly  
             > 365  : daily  
- method  : Any
```

Optional input:

```
- 'interpolateDate' : The input after this input must either be  
                      'start' or 'end'.
```

Date of interpolation. Where 'start' means
to interpolate the start date of the
periods of the old frequency, while 'end'
uses the end date of the periods.

Default is 'start'.

Caution : Only when converting to higher
frequency.

Output:

```
- obj : An nb_dateInExpr object.
```

Written by Kenneth S. Paulsen

► **corr ↑**

```
obj = corr(obj)
```

Description:

Correlation

Input:

```
- obj : An object of class nb_objectInExpr
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cos** ↑

```
obj = cos(obj)
```

Description:

Cosine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

cosd(obj) is the cosine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = cosd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cosh** ↑

obj = cosh(obj)

Description:

cosh(obj) is the hyperbolic cosine.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = cosh(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cot** ↑

obj = cot(obj)

Description:

Cotangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = cot(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

`cotd(obj)` is the cotangent of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = cotd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **coth** ↑

```
obj = coth(obj)
```

Description:

`coth(obj)` is the hyperbolic cotangent of the elements of `obj`.

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = coth(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **cov** ↑

```
obj = cov(obj)
```

Description:

Covariance

Input:

```
obj : An object of class nb_objectInExpr
```

Output:

```
d : An object of class nb_objectInExpr
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **csc** ↑

```
obj = csc(obj)
```

Description:

Cosecant of argument in radians.

Input:

```
- obj : An object of class nb_objectInExpr
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = cscd(in);
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **csch** ↑

```
obj = csch(obj)
```

Description:

csch(obj) is the hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = csch(in);
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cumprod** ↑

```
obj = cumprod(obj, dim)
```

Description:

Cumulativ product

Input:

- obj : An object of class nb_objectInExpr
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cumprod(obj,1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum

Input:

- obj : An object of class nb_objectInExpr
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cumsum(obj,1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **demean** ↑

```
obj = demean(data,~)
```

Description:

- Demeans along the dimension you choose.

Input:

- obj : A nb_objectInExpr object
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_objectInExpr object.

See also:

[sgrowth](#), [subAvg](#).

Written by Tobias Ingebrigtsen

Inherited from superclass NB_OBJECTINEXPR

► **demeanMoving** ↑

```
obj = demeanMoving(obj,backward)
obj = demeanMoving(obj,backward,forward)
```

Description:

- Demeans data with a moving average process.

Input:

- obj : A nb_objectInExpr object
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average

Output:

- obj : A nb_objectInExpr object.

See also:

[nb_dateInExpr.demean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **denton** ↑

obj = denton(obj,z,k,type,d)

Description:

The Denton method of transforming a series from low to high frequency.

See Denton (1971), Adjustment of Monthly or Quarterly Series to Annual Totals: An Approach Based on Quadratic Minimization.

Input:

- obj : A nb_dateInExpr object.
- z : Any
- k : The number intra low frequency observations. If you have annual data and want out quarterly data use 4.
- type : Any
- d : Any

Output:

- x : A nb_dateInExpr object.

See also:

[nb_ts.convert](#), [nb_denton](#)

Written by Kenneth Sæterhagen Paulsen

► **deptcat** ↑

obj = deptcat(a,b,varargin)

Description:

Depth concatenation (add pages of different nb_objectInExpr objects)
(No short hand notation)

Input:

```
- a : An object of class nb_objectInExpr  
- b : An object of class nb_objectInExpr  
- varargin : Optional number of nb_objectInExpr objects
```

Output:

```
obj : An object of class nb_objectInExpr.
```

Examples:

```
obj = deptcat(a,b);  
obj = deptcat(a,b,c);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **detrend** ↑

```
obj = detrend(obj,method,output,varargin)
```

Description:

De-trend.

Input:

```
- obj : A nb_objectInExpr object.  
- method : Any  
- output : Any
```

Optional input:

Any

Output:

```
- obj : A nb_objectInExpr object.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **diff** ↑

```
obj = diff(obj)
obj = diff(obj,lag)
obj = diff(obj,lag,skipNaN)
```

Description:

Diff method.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- skipNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = diff(obj);
obj = diff(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **disp ↑**

```
disp(obj)
```

Description:

Display object (On the commandline)

Input:

obj : An object of class nb_dateInExpr

Output:

The object display on the command line. (When not ending the command with an semicolon)

Examples:

```
obj
```

Written by Kenneth S. Paulsen

► **egrowth** ↑

```
obj = egrowth(obj,lag,stripNaN)
```

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = egrowth(obj);  
obj = egrowth(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **epcn** ↑

```
obj = epcn(obj,lag,stripNaN)
```

Description:

Calculate exact percentage growth, using the formula:
 $100 * (x(t) - x(t-1))/x(t-1)$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = epcn(obj); % period-on-period growth  
obj = epcn(obj,4); % 4-periods growth
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **eq** ↑

```
a = eq(a,b)
```

Description:

This method is intended to test if two nb_math_ts objects are equal. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar.
- b : An object of class nb_objectInExpr or a scalar.

Output:

- a : An object of class nb_objectInExpr

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **exp** ↑

```
obj = exp(obj)
```

Description:

Exponential function.

Input:

- obj : An object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr.

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **expand** ↑

```
obj = expand(obj,newStartDate,newEndDate,type,warningOff)
```

Description:

This method expands a nb_math_ts object, which means we need to take max of existing end date and the new end date.

Input:

- obj : An object of class nb_dateInExpr
- newStartDate : Any
- newEndDate : The wanted new end date of the data.
- type : Any
- warningOff : Any

Output:

- obj : An nb_dateInExpr object.

Examples:

```
obj = expand(obj,'1900Q1','2050Q1');  
obj = expand(obj,'1900Q1','2050Q1','zeros');
```

Written by Kenneth S. Paulsen

► **expml** ↑

```
obj = expml(obj)
```

Description:

```
expml(obj) computes exp(obj)-1, compensating for the roundoff in  
exp(obj).  
(For small real X, expml(X) should be approximately X, whereas the  
computed value of EXP(X)-1 can be zero or have high relative error.)
```

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = expml(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **extrapolate** ↑

```
obj = extrapolate(obj,toDate)  
obj = extrapolate(obj,toDate,varargin)
```

Description:

Extrapolate the time-series.

Input:

- obj : An object of class nb_dateInExpr
- toDate : Extrapolate to the given date. If the date is before the end date of the given variable no changes will be made, and with no warning. Can also be a integer with the number of periods to extrapolate.

Optional inputs:

- Any

Output:

- obj : An nb_dateInExpr object.

Examples:

```
obj = nb_dateInExpr('2000M1');
obj = extrapolate(obj,'2004M3');
obj = extrapolate(obj,'2004M3','method','ar');
obj = extrapolate(obj,'low','method','ar','freq',4);
```

Written by Kenneth S. Paulsen

► **fillNaN** ↑

```
obj = fillNaN(obj)
obj = fillNaN(obj,date)
```

Description:

Fill in for nan values with last valid observation.

Input:

- obj : An object of class nb_dateInExpr.
- date : The end date of the returned dataset. Either a nb_date object or string with the date, or a number indicating how many periods back in time from the date today, i.e. 0 is today, -1 is past period and 1 is next period. Default is to use the end date of the object.

Caution : If the given date is before the date of the object nothing will be done to the returned object.

Output:

- obj : An object of class nb_dateInExpr.

Examples:

```
d2 = fillNaN(d);
```

Written by Kenneth Sæterhagen Paulsen

► **floor** ↑

```
obj = floor(obj)
```

Description:

floor(obj) rounds the data elements of obj to the nearest integers towards minus infinity

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = floor(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ge** ↑

```
a = ge(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is greater than or equal to another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **growth** ↑

```
obj = growth(obj)  
obj = growth(obj,lag,stripNaN)
```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = growth(obj);
obj = growth(obj, 4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► gt ↑

```
a = gt(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is greater than another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a > b;
obj = 2 > b;
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **h21** ↑

```
obj = h21(obj,freq,type)
```

Description:

Map high 2 low frequency observation with converting the frequency of the object.

Input:

- obj : An object of class nb_objectInExpr
- freq : Any
- type : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **horzcat** ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation ([a,b])

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional number of nb_objectInExpr objects

Output:

obj : An object of class nb_objectInExpr

Examples:

```
obj = [a,b];  
obj = [a,b,c];
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **horzcatfast** ↑

```
obj = horzcatfast(varargin)
```

Description:

Horizontal concatenation

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional number of nb_objectInExpr objects

Output:

```
obj : An object of class nb_objectInExpr
```

See also

`nb_dateInExpr.horzcat`

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **hpfilter** ↑

```
obj = hpfilter(obj,lambda)
```

Description:

HP-filtering.

Input:

- obj : An object of class nb_objectInExpr
- lambda : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
gap = hpfilter(data,3000);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **hpfilter1s** ↑

```
obj = hpfilter1s(obj,lambda)
```

Description:

Do one-sided hp-filtering.

Input:

- obj : An object of class nb_objectInExpr
- lambda : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
gap = hpfilter1s(data,3000)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iegrowth** ↑

```
obj = iegrowth(obj,initialValues,periods)
```

Description:

Inverse of exact growth, i.e. the inverse method of the egrowth method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = iegrowth(obj);
obj = iegrowth(obj,100);
obj = iegrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iegrowthnan** ↑

```
obj = iegrowthnan(obj)
obj = iegrowthnan(obj,initialValues,periods)
```

Description:

Inverse of exact growth, i.e. the inverse method of the egrowth method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iepcn** ↑

```
obj = iepcn(obj,initialValues,periods)
```

Description:

Inverse of exact percentage growth, i.e. the inverse method of the epcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = iepcn(obj);
obj = iepcn(obj,100);
obj = iepcn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iepcnnan** ↑

```
obj = iepcnnan(obj)
obj = iepcnnan(obj,initialValues,periods)
```

Description:

Inverse of exact percentage growth, i.e. the inverse method of the epcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **igrowth** ↑

```
obj = igrowth(obj,initialValues,periods)
```

Description:

Inverse of log approx. growth, i.e. the inverse method of the growth method

Input:

```
- obj : An object of class nb_objectInExpr  
- InitialValues : Any  
- periods : Any
```

Output:

```
- obj : An nb_objectInExpr object.
```

Examples:

```
obj = igrowth(obj);  
obj = igrowth(obj,100);  
obj = igrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **igrowthnan** ↑

```
obj = igrowthnan(obj)  
obj = igrowthnan(obj,initialValues,periods)
```

Description:

Inverse of log approx. growth, i.e. the inverse method of the growth method

Input:

```
- obj : An object of class nb_objectInExpr  
- InitialValues : Any  
- periods : Any
```

Output:

```
- obj : An nb_objectInExpr object.
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate.

Input:

- data : A nb_objectInExpr object.
- method : Any

Output:

- data : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ipcn** ↑

```
obj = ipcn(obj)
obj = ipcn(obj,initialValues,periods,startAtNaN)
```

Description:

Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any
- startAtNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = ipcn(obj);
obj = ipcn(obj,100);
obj = ipcn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ipcnnan** ↑

```
obj = ipcnnan(obj)
obj = ipcnnan(obj,initialValues,periods)
```

Description:

Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Is finite.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Is nan.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **kurtosis** ↑

```
obj = kurtosis(obj,flag,dim,outputType)
```

Description:

Calculate the kurtosis.

Input:

- obj : An object of class nb_objectInExpr

- flag : Any

- dim : Any

- outputType : - 'nb_math_ts': The result will be an object
of class nb_objectInExpr.

- 'double': The result will be an empty object
of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **lag** ↑

obj = lag(obj,periods)

Description:

Lag method.

Input:

- obj : An object of class nb_objectInExpr

- periods : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = lag(obj,1);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ldivide** ↑

obj = ldivide(a,b)

Description:

Left element-wise division (.\ \backslash). Same as right element-wise (./) division. See rdivide for more.

Input:

- a : An object of class nb_objectInExpr or a scalar

- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a.\b;  
obj = 2.\b;  
obj = a.\2;
```

See also:

[rdivide](#), [mldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **le ↑**

```
a = le(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is less than or equal to another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **lead ↑**

```
obj = lead(obj,periods)
```

Description:

Lead method.

Input:

- obj : An object of class nb_objectInExpr
- periods : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = lead(obj,1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log ↑**

```
obj = log(obj)
```

Description:

Log operator

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log10 ↑**

```
obj = log10(obj)
```

Description:

Take the common (base 10) log

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = log1p(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log2** ↑

```
obj = log2(obj)
```

Description:

log2(obj) is the base 2 logarithm of the elements of obj.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log2(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **lt ↑**

```
a = lt(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is less than another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a < b;
obj = 2 < b;
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mavg** ↑

```
obj = mavg(obj,backward,forward,flag)
```

Description:

Taking moving avarage

Input:

- obj : An object of class nb_objectInExpr
- backward : Any
- forward : Any
- flag : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **max** ↑

```
obj = max(obj,dim,outputType)
```

Description:

Max

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mean** ↑

```
obj = mean(obj,dim,outputType)
```

Description:

Mean

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **median** ↑

```
obj = median(obj,dim,outputType)
```

Description:

Median.

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **merge2Series** ↑

obj = merge2Series(obj,other,method,varargin)

Description:

Merge 2 series.

Input:

- obj : An object of class nb_objectInExpr.

- other : An object of class nb_objectInExpr.

- method : Any

Optional input:

- Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **min** ↑

obj = min(obj,dim,outputType)

Description:

Min

Input:

- obj : An object of class nb_objectInExpr

- dim : Any

- `outputType` : - 'nb_math_ts': The result will be an object of class `nb_objectInExpr`.
 - 'double': The result will be an empty object of class `nb_objectInExpr`.

Output:

- `obj` : An object of class `nb_objectInExpr`

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **minus** ↑

`obj = minus(a,b)`

Description:

Binary subtraction (-).

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar
- `b` : An object of class `nb_objectInExpr` or a scalar

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
obj = a-b;
obj = 2-b;
obj = a-2;
```

See also:

[plus](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **mldivide** ↑

`obj = mldivide(a,b)`

Description:

Matrix left division (\).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = 2\obj;  
obj = obj\2;
```

See also:

[rdivide](#), [ldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mpower** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^)

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = 2^b;  
obj = a^2;
```

See also:

[power](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **mrdivide** ↑

`obj = mrdivide(a,b)`

Description:

Matrix left division (/).

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar
- `b` : An object of class `nb_objectInExpr` or a scalar

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
obj = 2/obj;  
obj = obj/2;
```

See also:

[rdivide](#), [ldivide](#), [mldivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **mstd** ↑

`obj = mstd(obj,backward,forward)`

Description:

Moving standard deviation

Input:

- obj : An object of class nb_objectInExpr
- backward : Number of periods backward in time to calculate the moving std
- forward : Number of periods forward in time to calculate the moving std

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mtimes** ↑

obj = power(a,b)

Description:

Matrix multiplication (*).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = 2*b;
obj = a*2;

See also:

[times](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ne** ↑

a = ne(a,b)

Description:

This method is intended to test if two nb_math_ts objects are not equal. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar.
- b : An object of class nb_objectInExpr or a scalar.

Output:

- a : An object of class nb_objectInExpr

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **not** ↑

```
obj = not(obj)
```

Description:

The not operator (~)

Input:

- a : An object of class nb_objectInExpr

Output:

- a : An object of class nb_objectInExpr

Examples:

```
obj = ~obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **or** ↑

```
a = or(a,b)
```

Description

The `or` operator (`||`)

Input:

- `a` : An object of class `nb_objectInExpr`
- `b` : An object of class `nb_objectInExpr`

Output:

- `a` : An object of class `nb_objectInExpr`

Examples:

```
a = a | b;
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **pca ↑**

```
F = pca(obj)
[F,LAMBDA,R,varF,expl,c,sigma,e,Z] = pca(obj,r,method,varargin)
```

Description:

Principal Component Analysis.

Input:

- `obj` : An object of class `nb_objectInExpr`.
- The rest not important

Output:

- `F` : A `nb_objectInExpr` object.
- `LAMBDA` : A empty `nb_objectInExpr` object.
- `R` : A empty `nb_objectInExpr` object.
- `varF` : A empty `nb_objectInExpr` object.
- `expl` : A empty `nb_objectInExpr` object.
- `c` : A empty `nb_objectInExpr` object.

- sigma : A empty nb_objectInExpr object.
- e : A nb_objectInExpr object.
- Z : A nb_objectInExpr object.

See also:

[nb_pca](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pca_func** ↑

factors = pca_func(varargin)

Description:

Principal Component of selected number of series represented by a set of nb_objectInExpr.

Optional input:

- varargin : Optional number of nb_objectInExpr objects.
- 'numFactors' : Determine number of Factors to report. Standard is first factor only.
- 'unbalanced' : true or false.
- 'whichFactor' : Select the factor to return.

Output:

- factors : An object of class nb_objectInExpr.

See also:

[nb_dateInExpr.pca](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pcn** ↑

obj = pcn(obj,lag,stripNaN)

Description:

Calculate log approximated percentage growth. Using the formula
 $(\log(t+lag) - \log(t)) * 100$

Input:

- obj : An object of class nb_objectInExpr
- lag : The number of lags. Default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = pcn(obj); % period-on-period log approx. growth
obj = pcn(obj,4); % 4-periods log approx. growth
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **plus ↑**

```
obj = plus(a,b)
```

Description:

Binary addition (+).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a+b;
obj = 2+b;
obj = a+2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pow2** ↑

```
obj = pow2(obj)
```

Description:

pow2(obj) raises the number 2 to the object

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = pow2(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **power** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^)

Input:

- a : An object of class nb_objectInExpr or a scalar

- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a.^b;
obj = 2.^b;
obj = a.^2;
```

See also:

[mpower](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **q2y** [↑](#)

`obj = q2y(obj)`

Description:

Cumulative sum over the last 4 periods.

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

`obj = q2y(obj)`

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **rdivide** [↑](#)

`obj = rdivide(a,b)`

Description:

Right element-wise division (`./`)

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar
- `b` : An object of class `nb_objectInExpr` or a scalar

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
obj = a./b;  
obj = 2./b;  
obj = a./2;
```

See also:

[mrdivide](#), [mldivide](#), [ldivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **reIndex** ↑

```
obj = reIndex(obj,date)
```

Description:

Reindex the data of the object to a new date.

Input:

- obj : An object of class nb_objectInExpr
- date : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **real** ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **reallog** ↑

obj = reallog(obj)

Description:

The real natural logarithm

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = reallog(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **realpow** ↑

obj = realpow(a,b)

Description:

Real power.

Input:

- a : An object of class nb_objectInExpr or a scalar

- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

See also:

mpower, power

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **realsqrt** ↑

obj = realsqrt(obj)

Description:

realsqrt(obj) is the the real square root

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = realsqrt(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ret** ↑

obj = ret(obj)
obj = ret(obj,nlag)
obj = ret(obj,nlag,skipNaN)

Description:

Calculate return, using the formula: $x(t)/x(t-lag)$.

Input:

- obj : An object of class nb_objectInExpr
- nlag : Any
- skipNaN : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = ret(obj);  
obj = ret(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **rlag** ↑

```
obj = rlag(obj,lags)  
obj = rlag(obj,lags,periods)
```

Description:

Roll a pattern in the data forward with a given lag.

Input:

- obj : An object of class nb_objectInExpr
- lags : Any
- periods : The extrapolated periods. The end date of the object will be the current end date plus these number of periods. Be aware that trailing nan values in the data will be filled in for even if periods == 0, which is the default.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **round** ↑

```
obj = round(obj,value,startDate,endDate,pages)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

```
- obj : A nb_objectInExpr object  
- value : Any  
- startDate : Any  
- endDate : Any  
- variables : Any
```

Output:

```
- obj : A nb_objectInExpr object
```

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sec ↑**

```
obj = sec(obj)
```

Description:

Secant of argument in radians.

Input:

```
- obj : An object of class nb_objectInExpr
```

Output:

```
- obj : An object of class nb_objectInExpr.
```

Examples:

```
obj = sec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **secd ↑**

```
obj = secd(obj)
```

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = secd(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sech** ↑

obj = sech(obj)

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = sech(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **set2Value** ↑

obj = set2Value(obj, func, value)

Description:

Set all values applying to the restriction given by the input func to the provided value.

Input:

- obj : An object of class nb_objectInExpr.
- func : Any
- value : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► setNan2Zero ↑

obj = setNan2Zero(obj)

Description:

Set all nan values of the data of the object to 0.

Input:

- obj : An object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► sgrowth ↑

obj = sgrowth(obj,horizon)

Description:

Calculates smoothed 12 or 6 months growth.

$x(*) = ((x13+x14+x15)/(x1+x2+x3)-1)*100$

$x(*) = ((x7+x8+x9)/(x1+x2+x3)-1)*100$

Input:

- obj : A nb_objectInExpr object
- horizon : Any

Output:

- out : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sin** ↑

obj = sin(obj)

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr.

Examples:

obj = sin(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sind** ↑

obj = sind(obj)

Description:

Sine of argument in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sind(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► sinh ↑

```
obj = sinh(obj)
```

Description:

Hyperbolic sine.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sinh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► skewness ↑

```
obj = skewness(obj,flag,dim,outputType)
```

Description:

Calculate the skewness

Input:

- obj : An object of class nb_objectInExpr

- flag : Any

- dim : Any

- outputType : - 'nb_math_ts': The result will be an object
of class nb_objectInExpr.

- 'double': The result will be an empty object
of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = sqrt(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **std** ↑

obj = std(obj,flag,dim,outputType)

Description:

Standard deviation

Input:

- obj : An object of class nb_objectInExpr

- flag : Any

- dim : Any

- outputType : - 'nb_math_ts': The result will be an object
of class nb_objectInExpr.

- 'double': The result will be an empty object
of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **stdise** ↑

obj = stdise(obj,flag)

Description:

Standardise

Input :

- obj : An object of class nb_objectInExpr

- flag : Any

Output:

- obj : An nb_objectInExpr object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **subAvg** ↑

obj = subAvg(obj,k,w)

Description:

- Will for the last k periods (including the present) calculate the cumulative sum and then divide by the amount of periods, which gives you the average over those periods.

Input:

- obj : An object of class nb_objectInExpr.

- k : Any

- w : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **subSum** ↑

obj = subSum(obj, k, w)

Description:

Calculates the cumulative sum over the last k periods (including the present period).

Input:

- obj : An object of class nb_objectInExpr.
- k : Any
- w : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sum** ↑

obj = sum(obj, dim)

Description:

Takes the sum of the object data in the wanted dimension

Input:

- obj : An object of class nb_objectInExpr
- dim : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = sum(obj,1)
obj = sum(obj,2)
obj = sum(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sumOAF** ↑

```
obj = sumOAF(obj,sumFreq)
```

Description:

Take the sum over a lower frequency.

Input:

- obj : An object of class nb_objectInExpr
- sumFreq : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **tan** ↑

```
obj = tan(obj)
```

Description:

Tangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr.

Examples:

```
obj = tan(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_OBJECTINEXPR
```

► **tand** ↑

```
obj = tand(obj)
```

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = tand(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

Hyperbolic tangent.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = tanh(in);  
  
Written by Kenneth Sæterhagen Paulsen  
  
Inherited from superclass NB_OBJECTINEXPR
```

► **times** ↑

```
obj = times(a,b)
```

Description:

Element-wise multiplication (.*)

Input:

- a : An object of class nb_objectInExpr or a scalar double
- b : An object of class nb_objectInExpr or a scalar double

Output:

- obj : An object of class nb_objectInExpr

See also:

[mtimes](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **uminus** ↑

```
obj = uminus(obj)
```

Description

Unary minus

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = -obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR
```

► **undiff** ↑

```
obj = undiff(obj,initialValues,periods)
```

Description:

Inverse of the diff method.

Input:

- obj : An object of class nb_objectInExpr.
- initialValues : Any.
- periods : Any.

Output:

- Output : An object of class nb_objectInExpr.

See also
nb_dateInExpr.diff

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **var** ↑

```
obj = var(obj,dim,outputType)
```

Description:

Variance

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **vertcat** ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional numbers of objects of class nb_objectInExpr

Output:

- a : An nb_objectInExpr object

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **window** ↑

```
obj = window(obj,startDateWin,endDateWin, pages)
```

Description:

Narrow down window

Input:

- obj : An object of class nb_dateInExpr

- startDateWin : Any

- endDateWin : The new wanted end date of the data of the object. Must be a string on the format. 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the end date of the data.

Can also be an object which is a subclass of the nb_date class.

- pages : Any

Output:

- obj : An object of class nb_dateInExpr

Written by Kenneth S. Paulsen

► **x12** ↑

```
obj = x12(obj,range,varargin)
```

Description:

Seasonally adjust

Input:

```
- obj : An object of class nb_objectInExpr.  
- range : Any  
  
Optional input (..., 'propertyName', propertyName, ...):  
Any
```

Output:

```
- obj : An object of class nb_objectInExpr.
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **x12Census** ↑

```
obj = x12Census(obj)  
[obj,x,outputfile,errorfile,model] = x12Census(obj)  
[obj,x,outputfile,errorfile,model] = x12Census(obj,varargin)
```

Description:

Do x12 Census

Input:

```
- obj : An object of class nb_objectInExpr.  
  
Optional input (..., 'propertyName', propertyName, ...):  
  
Any.
```

Output:

```
- obj : An object of class nb_objectInExpr.  
- x : An object of class nb_objectInExpr.  
- outputfile : ''  
- errorfile : {}  
- mdl : []
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

■ **nb_day**
Go to: [Properties](#) | [Methods](#)

A class representing a daily date.

Superclasses:

`nb_date`

Constructor:

```
obj = nb_day(day)
obj = nb_day(day,month,year)
```

Input:

- day : A string on one of the following date formats:

```
> 'yyyyMm(m)Dd(d)', e.g. '2012M1D10'
> 'dd.mm.yyyy', e.g. '10.01.2012'
> 'ddmonyyyy', e.g. '10jan2012'
> 'yyyymmdd', e.g. '20121231'
```

Or an integer with the day. (Must be combined with the month and year arguments)

- month : Must be an integer with the month.

- year : Must be an integer with the year.

Output:

- obj : An `nb_day` object.

Examples:

```
obj = nb_day('2012M1D10');
obj = nb_day(1,1,2012);
```

See also:

[nb_date](#), [nb_year](#), [nb_semiAnnual](#), [nb_quarter](#), [nb_month](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- `baseYear`
- `day`
- `dayNr`
- `dayOfWeek`
- `frequency`
- `leapYear`
- `month`
- `quarter`
- `year`

- **baseYear** ↑

The base year. 2000.

Inherited from superclass `NB_DATE`

- **day** ↑

The given day as a double

- **dayNr** ↑

The number of days since the base year, which is 1.1.2000

- **dayOfWeek** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass NB_DATE

- **frequency** ↑

The frequency of the date. Must be 365.

- **leapYear** ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass NB_DATE

- **month** ↑

The month of the given day

- **quarter** ↑

The quarter of the given day

- **year** ↑

The year of the given day

Methods:

- ascensionDay
 - colon
 - convert
 - date2freq
 - disp
 - easter
 - eq
 - format2string
 - freqPlus
 - fromDate
 - ge
 - getDate
 - getEarliestDate
 - getFreq
 - getFrequencyAsInteger
 - getFrequencyAsString
 - getHalfYear
 - getLatestDate
 - getMonth
 - getNr
 - getNumberOfDaysInMonth
 - getNumberOfDaysUntilNowInMonth
 - getNumberOfDaysUntilNowInYear
 - getQuarter
 - getWeek
 - getWeekNumber
 - getYear
 - gt
 - holidays
 - initialize
 - intersect
 - isDate
 - isEqualFreq
 - isFirstPeriod
 - isLeapYear
 - isPeriod
 - isempty
 - isequal
 - ismember
 - le
 - lt
 - max
 - min
 - minus
 - ne
 - pentecost
 - plus
 - setDayOfWeek
 - sort
 - toDate
 - toDates
 - toFormat

 - toString
 - today
 - union
 - unique
 - vec
 - vintage2Date
 - weekday
-

► **ascensionDay** ↑

```
day = ascensionDay(obj,~)
```

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_day.easter](#), [nb_day.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **cell2Date** ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.

- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'
      '2011Q3'
      '2011Q4'
      '2012Q1'
      '2012Q2'
```

```
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'}  
  
a = nb_date.cell2Date(g,4)  
  
a =  
  
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_day.ascensionDay](#), [nb_day.pentecost](#), [nb_day.easter](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a,d,b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► convert ↑

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_day object ot the wanted frequency. The return variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_day
- frequency : The frequency to convert to. As a scalar.
- first : Does nothing. Just to make the code general for all frequencies.

Output:

- obj : An object of subclass of the nb_date class with the frequency given by the frequency input.

Examples:

```
obj = nb_day(1,1,2012);
obj = obj.convert(1);
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **date2freq ↑**

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'
semiannually : 'yyyySs' or 'dd.mm.yyyy'
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
> otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
> 1 : yearly

```
> 2    : semiannually  
> 4    : quarterly  
> 12   : monthly  
> 52   : weekly  
> 365  : daily  
  
- type      : 1 if the dates input is on the xls format,  
               otherwise 0
```

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp** ↑

```
disp(obj)
```

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

```
day = easter(obj,type)
```

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- obj : An object of class nb_date.
- type : 'all' or 'sunday' (default).

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_day.ascensionDay](#), [nb_day.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **eq** ↑

```
ret = eq(obj,aObj)
```

Description:

Tests if the two objects represent the same dates

Input:

- obj : An object of class nb_day
- aObj : Another object of class nb_day

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj,format)
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- d : Day of the date. Examples:

```
> 'dd'   : '01', '10'  
> 'd'    : '1', '0'  
> 'd(d)': '1', '10'  
- w : Week of date. Same options as for d.  
- m : Month of date. Same options as for d.
```

- q : Quarter of date. Examples:

```
> 'q'   : '1', '4'  
- h : Half year of date. Same options as for h.  
- y : Year of date.  
    > 'yyyy' : '2016'  
    > 'yy'   : '16'  
    < 'y'    : '6'
```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'
 > 'english' : 'Q'
- 'Weektext' : > 'norwegian' : 'Uke'
 > 'english' : 'Week'
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d      = nb_day.today();  
date = nb_date.format2string(d,'dd.mm.yyyy');
```

```

date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m     = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q     = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y     = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');

```

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► [freqPlus ↑](#)

obj = freqPlus(obj,incr,freq)

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **ge** ↑

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater than or equal to another object.
Which will mean that the first input is after or the same
as the second input in the calendar

Input:

- obj : An object of class nb_day
- aObj : Another object of class nb_day

Output:

- ret : 1 if obj >= aObj, 0 else

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate** ↑

```
date = getDate(obj, freq)
```

Description:

Get the date of the object at another frequency.

Input:

- obj : An object of class nb_day
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(4); % Will return a nb_quarter object
```

See also:

[nb_day.convert](#)

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getFreq** ↑

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

- obj : An object of class nb_day

Output:

- freq : 365

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger** ↑

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Output:

- frequency : The frequency as a string

> Daily	:	'daily'
> Weekly	:	'weekly'
> Monthly	:	'monthly'
> Quarterly	:	'quarterly'
> Semiannually	:	'semiannually'
> Yearly	:	'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getHalfYear** ↑

```
halfYear = getHalfYear(obj)
```

Description:

```
Get the half year of the day, given as an nb_semiAnnual object
```

Input:

- obj : An object of class nb_day

Output:

- halfYear : An nb_semiAnnual object

Examples:

```
halfYear = obj.getHalfYear(); % Will return the half year
```

Written by Kenneth S. Paulsen

► **getLatestDate** ↑

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATE

► **getMonth** ↑

```
month = getMonth(obj)
```

Description:

Get the month of the day, given as a nb_month object

Input:

- obj : An object of class nb_day

Output:

- month : An nb_month object

Examples:

```
month = obj.getMonth(); % Will return the month
```

Written by Kenneth S. Paulsen

► **getNr** ↑

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDaysInMonth** ↑

```
numberOfDays = nb_day.getNumberOfDaysInMonth(month, leapYear)
```

Description:

A static method of the class nb_day.

Get the number of days in the given month

Input:

```
- month      : An object of the class nb_day  
- leapYear  : 1 if leapYear, 0 else
```

Output:

```
- numberOfDaysInMonth : Returns the number of days in the current  
month
```

Examples:

```
numberOfDays = nb_day.getNumberOfDaysInMonth(2,1); % returns 29  
numberOfDays = nb_day.getNumberOfDaysInMonth(2,0); % returns 28  
numberOfDays = nb_day.getNumberOfDaysInMonth(3,1); % returns 31
```

Written by Kenneth S. Paulsen

► **getNumberOfDaysUntilNowInMonth ↑**

```
numberOfDaysUntilNowInMonth = getNumberOfDaysUntilNowInMonth(obj)
```

Description:

Get the number of days in the current month

Input:

```
- obj : An object of class nb_day
```

Output:

```
- numberOfDaysUntilNowInMonth : Number of days of the current  
month of the date object.
```

Examples:

```
ret = obj.getNumberOfDaysUntilNowInMonth();
```

Written by Kenneth S. Paulsen

► **getNumberOfDaysUntilNowInYear ↑**

```
numberOfDaysUntilNowInYear = getNumberOfDaysUntilNowInYear(obj)
```

Description:

Get the number of days until now in the current year

Input:

- obj : An nb_day object

Output:

- `numberOfDaysUntilNowInYear` : Number of days until now in the current year as a double.

Examples:

```
ret = obj.getNumberOfDaysUntilNowInYear();
```

Written by Kenneth S. Paulsen

► **getQuarter** ↑

```
quarter = getQuarter(obj)
```

Description:

Get the quarter of the day, given as a nb_quarter object

Input:

- obj : An object of class nb_day

Output:

- quarter : An nb_quarter object

Examples:

```
quarter = obj.getQuarter(); % Will return the current quarter
```

Written by Kenneth S. Paulsen

► **getWeek** ↑

```
week = getWeek(obj,dayOfWeek)
```

Description:

Get the week of the day, given as an nb_week object

Input:

- obj : An object of class nb_day
- dayOfWeek : The weekday the given week represents when converted to a day. (1-7 (Monday-Sunday)). Default is to use the weekday(obj).

Output:

- week : An nb_week object

Examples:

```
week = obj.getWeek();      % Will return the week of the day
```

Written by Kenneth S. Paulsen

► **getWeekNumber ↑**

```
[weekNum,weekDay,yearN] = getWeekNumber(obj)
```

Description:

Get week number and weekday of a nb_day object.

Input:

- obj : A nb_day object.

Output:

- weekNum : The week number that the day is in. As a double.
- weekDay : The weekday of the day. As an integer. Sun: 1, Mon: 2, Tue: 3, Wed: 4, Thu: 5, Fri: 6, Sat: 7.
- yearN : The year of the day, as a double.

Written by Kenneth Sæterhagen Paulsen

► **getYear ↑**

```
year = getYear(obj)
```

Description:

Get the year of the day, given as a nb_year object

Input:

- obj : An object of class nb_day

Output:

- year : A nb_Year object

Examples:

```
day = nb_day(1,1,2020);
year = day.getYear(); % Will return the current year
```

Written by Kenneth S. Paulsen

► gt ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater than another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_day
- aObj : An object of class nb_day

Output:

- ret : 1 if obj > aObj, 0 else

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► holidays ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_day.easter](#), [nb_day.ascensionDay](#), [nb_day.pentecost](#)
[nb_day.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize** ↑

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **intersect** ↑

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **isDate** ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **isEqualFreq** ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod** ↑

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)

m = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isLeapYear** ↑

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE
```

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_day

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = obj.isempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : An object of class nb_day
- aObj : Another object of class nb_day

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

ret = le(obj,aObj)

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

- obj : An object of class nb_day
- aObj : An object of class nb_day

Output:

- ret : 1 if obj <= aObj, 0 else

Examples:

ret = obj <= aObj;

Written by Kenneth S. Paulsen

► **lt** ↑

ret = lt(obj,aObj)

Description:

Tests if an object is less than another object. Which will mean that the first input is before the second input in the calendar

Input:

- obj : An object of class nb_day
- aObj : An object of class nb_day

Output:

- ret : 1 if obj < aObj, 0 else

Examples:

ret = obj < aObj;

Written by Kenneth S. Paulsen

► **max** ↑

d = max(obj)
[d,ind] = max(obj)
d = max(obj,aObj)

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_day objects.

Two inputs:

- obj : A scalar nb_day object.

- aObj : A scalar nb_day object.

Output:

- d : A scalar nb_day object.

- ind : The index of the last date.

Written by Kenneth Sæterhagen Paulsen

► **min ↑**

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_day objects.

Two inputs:

- obj : A scalar nb_day object.

- aObj : A scalar nb_day object.

Output:

- d : A scalar nb_day object.

- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► **minus ↑**

```
output = minus(obj,aObj)
```

Description:

Makes it possible to take an nb_day object minus another nb_day object and get the number of days between them.

It makes it also possible to take the difference between an nb_day object and a string one one of the following formats:
'yyyyMm(m)Dd(d)', 'dd.mm.yyyy'

And last it makes it possible to subtract the number of days from an nb_day object and receive an nb_day representing the day the given number of days backward in time.

Input:

- obj : A date string, an nb_day object or an integer.
- aObj : A date string, an nb_day object or an integer.

Output:

- output : The number of days between the given days. As a double

or

An nb_day object representing the date the given number of days backwards in time.

Examples:

```
output = obj - aObj;           % (both objects). Returns a double
output = obj - '2012M1D1';    % Returns a double
output = '2012M12D23' - obj;  % Returns a double
output = obj - 1;              % Returns an object
same as
output = 1 - obj;             % Returns an object
```

Written by Kenneth S. Paulsen

► **ne ↑**

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_day
- aObj : An object of class nb_day

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_day.easter](#), [nb_day.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

```
out = plus(obj,aObj)
```

Description:

Makes it possible to add a given number of days to an nb_day and receive a nb_day object which represents the day the given numbers of days ahead

Input:

- obj : A nx1 double or a nx1 nb_day object
- aObj : A nx1 double or a nx1 nb_day object

Output:

- out : A nx1 nb_day object with the given number of years ahead

Examples:

```
obj = obj + 1;
```

same as

```
obj = 1 + obj;
```

Written by Kenneth S. Paulsen

► **setDayOfWeek** ↑

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_day.dayOfWeek](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **sort** ↑

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toDate ↑**

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

- > Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'
- > Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'
- > Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'
- > Quarterly : 'dd.mm.yyyy' and 'yyyyQq'
- > Semiannually : 'dd.mm.yyyy' and 'yyyySs'
- > Yearly : 'dd.mm.yyyy' and 'yyyy'
- > Secondly : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss'

(only supported if this version of the toolbox
includes the nb_second class)

- frequency :

```
> Daily      : 365
> Weekly     : 52
> Monthly    : 12
> Quarterly  : 4
> Semiannually : 2
> Yearly     : 1
> Secondly   : 31536000
```

Output:

- dObj :

```
> Daily      : An nb_day object
> Weekly     : An nb_week object
> Monthly    : An nb_month object
> Quarterly  : An nb_quarter object
> Semiannually : An nb_semiAnnual object
> Yearly     : An nb_year object
> Secondly   : An nb_second object
```

Examples:

```
dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates ↑**

```
varargout = toDates(obj,periods,format,newFreq,first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : An object of class nb_day
- periods : The periods wanted, must be given as a double array.
E.g. 1:20.
- format : Set the format of the cellstr of dates:
 - > 'nb_date' : The output will be a vector of nb_date objects.
 - > 'default' : 'yyyyMm(m)Dd(d)'
 - > 'fame' : 'ddmonyyyy', e.g. '10jan2012'
 - > 'xls' : 'dd.mm.yyyy'
 - > 'vintage' : 'yyyymmdd'

Formats which is dependent on the 'newFreq' input

> 'NBNorsk' ('NBNorwegian'):

```
newFreq;  
> 1 : 'YYYY'  
> 2 : 'mmm. yy', e.g. 'jan. 08' and 'jul. 08'  
> 4 : 'q.kv.yy'  
> 12 : 'mmm. yy', e.g. mmm = 'jan'  
> 52 : 'Ukew(w)-yy'  
> 365 : 'dd.mm.yyyy'
```

(Date strings are only given for the days which ends the periods)

> 'NBEnglish' ('NBEngelsk'):

```
newFreq;  
> 1 : 'YYYY'  
> 2 : 'mmm-yy', e.g. 'jan. 08' and 'jul. 08'  
> 4 : 'yyyyQq'  
> 12 : 'mmm-yy', e.g. mmm = 'jan'  
> 52 : 'Weekw(w)-yy'  
> 365 : 'dd.mm.yyyy'
```

(Date strings are only given for the days which ends the periods)

- newFreq : > 1 : yearly
> 2 : semiannually
> 4 : quarterly
> 12 : monthly
> 365 : daily
- first : When converting to a lower frequency you must set if you want to use the first date or the last date as the location of the returned date. The default is to use the first.

```
1 = first  
0 = last
```

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates. Of interest when converting to lower frequencies

Examples:

```
obj          = nb_day(1,1,2000);  
dates        = obj.toDates(0:2000);  
dates        = obj.toDates(0:2000,'xls');  
dates        = obj.toDates(0:2000,'nb_date',12)  
[dates,locations] = obj.toDates(0:2000,'NBNorsk',1);
```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start,finish,format,language,first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string method.
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_day.format2string](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toString** ↑

```
date = toString(obj,format,first)
```

Description:

Transform the nb_day object to a string representing the date.

Input:

- obj : An object of class nb_day
- format :
 - > 'xls' : 'dd.mm.yyyy'
 - > 'pprnorsk' or 'mprnorwegian' : 'd(d). monthtext yyyy'
 - > 'pprengelsk' or 'mprenglish' : 'd(d) Monthtext yyyy'
 - > 'default' (otherwise) : 'yyyyMm(m)Dd(d)'
 - > 'vintage' : 'yyyymmdd'
- first :
 - 'xls' : If 1 is given, the first day of the current month will be returned. Default is not.

Output:

- date : A string with the date on the wanted format

Examples:

```
obj = nb_day(1,1,2012);
date = obj.toString(); % Will give '2012M1D1'
date = obj.toString('xls'); % Will give '01.01.2012'
date = obj.toString('xls',0); % Will give '01.01.2012'
date = obj.toString('nborsk'); % Will give '2012M1D1'
date = obj.toString('nbengelsk'); % Will give '2012M1D1'
date = obj.toString('pprnorsk'); % Will give '1. januar 2012'
date = obj.toString('pprnorsk',0); % Will give '1. januar 2012'
date = obj.toString('pprengelsk'); % Will give '1 January 2012'
```

Written by Kenneth S. Paulsen

► **today** ↑

```
obj = nb_day.today()
```

Description:

Get the current day as a nb_day object.

Output:

obj : An object of class nb_day.

Written by Kenneth SÃ¶terhagen Paulsen

► **union** ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

```
- out : A unique (sorted) vector of nb_date objects.  
- IA : out = in(IA)  
- IC : A = C(IC)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **vec** ↑

```
out = vec(obj,endD)
```

Description:

A method to create a vector of dates between two dates.

Input:

```
obj : A nb_day object which represents the start date of the vector  
end : A nb_day object or a double, representing the end date or the  
number of periods you want in your vector
```

Output:

```
obj : A nPeriods*1 vector of nb_day objects.
```

Examples:

```
f = nb_day(1,1,2010);  
g = nb_day(10,1,2010);  
vector = f.vec(g)
```

See also:

[nb_day](#)

Written by Tobias Ingebrigtsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |  
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► weekday ↑

```
[dayNum,dayName] = weekday(obj)
```

Description:

Get weekday of daily date.

Input:

- obj : An object of class nb_day.
- language : 'english' (default) or 'norwegian'.

Output:

- dayNum : As an integer. Sun: 1, Mon: 2, Tue: 3, Wed: 4, Thu: 5, Fri: 6, Sat: 7
- dayName : As a string. (A cellstr if numel(obj) > 1)

Written by Kenneth Sæterhagen Paulsen

■ nb_month

Go to: [Properties](#) | [Methods](#)

A class representing a monthly date.

Superclasses:

[nb_date](#)

Constructor:

`obj = nb_month(month,year)`

Input:

- month : A string on one of the following date formats:

> 'yyyyMm(m)'

> 'dd.mm.yyyy'

Or an integer with the month. (Must be combined with the year argument)

- year : Must be an integer with the year.

Output:

- obj : An nb_month object.

Examples:

```
obj = nb_month('2012M1');
obj = nb_month('01.01.2012');
obj = nb_month(1,2012);
```

See also:

[nb_date](#), [nb_year](#), [nb_semiAnnual](#), [nb_quarter](#), [nb_day](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [baseYear](#)
- [dayOfWeek](#)
- [frequency](#)
- [leapYear](#)
- [month](#)
- [monthNr](#)
- [quarter](#)
- [year](#)

- **baseYear** ↑

The base year. 2000.

Inherited from superclass NB_DATE

- **dayOfWeek** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass NB_DATE

- **frequency** ↑

The frequency of the date. Must be 12.

- **leapYear** ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass NB_DATE

- **month** ↑

The given month as a double

- **monthNr** ↑

The number of months since the base.
Which is the first month of 2000.

- **quarter** ↑

The given quarter as a double.

- **year** ↑

The given year as a double.

Methods:

- ascensionDay
- colon
- disp
- format2string
- ge
- getDays
- getFrequencyAsInteger
- getLatestDate
- getNumberOfWeeks
- getWeeks
- holidays
- isDate
- isLeapYear
- isequal
- lt
- ne
- setDayOfWeek
- toDates
- today
- vec
- cell2Date
- convert
- easter
- freqPlus
- getDate
- getEarliestDate
- getFrequencyAsString
- getNr
- getQuarter
- getYear
- initialize
- isEqualFreq
- isPeriod
- ismember
- max
- pentecost
- sort
- toFormat
- union
- vintage2Date
- christmas
- date2freq
- eq
- fromxlsDates2freq
- getDay
- getFreq
- getHalfYear
- getNumberOfDays
- getWeek
- gt
- intersect
- isFirstPeriod
- isempty
- le
- min
- plus
- toDate
- toString
- unique

► **ascensionDay** ↑

```
day = ascensionDay(obj,~)
```

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_month.easter](#), [nb_month.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **cell2Date** ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.

- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'  
     '2011Q3'  
     '2011Q4'  
     '2012Q1'  
     '2012Q2'  
     '2012Q3'  
     '2012Q4'  
     '2013Q1'  
     '2013Q2' }  
  
a = nb_date.cell2Date(g,4)
```

a =

```
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_month.ascensionDay](#), [nb_month.pentecost](#), [nb_month.easter](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a,d,b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;  
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **convert** ↑

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_month object ot the wanted frequency. The return variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_month
- frequency : The frequency to convert to. As a scalar.

```
- first      : 1 : first period  
              0 : last period  
  
Only when converting the object to a higher  
frequency.
```

Output:

```
- obj      : An object of subclass of the nb_date class with the  
frequency given by the frequency input.
```

Examples:

```
obj = nb_month(1,2012);  
obj = obj.convert(1);
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **date2freq** ↑

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this
function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'  
semiannually : 'yyyySs' or 'dd.mm.yyyy'  
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'  
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'  
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'  
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

```
- dates      : A cellstr of the dates you want to test the  
frequency of.  
  
- xls        : > 'xls' : If you want to test if the dates is on  
                  the format 'dd.mm.yyyy' also  
  
                  > otherwise will not (default)
```

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- type : 1 if the dates input is on the xls format, otherwise 0

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp** ↑

disp(obj)

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

```
day = easter(obj,type)
```

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- obj : An object of class nb_date.
- type : 'all' or 'sunday' (default).

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_month.ascensionDay](#), [nb_month.pentecost](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATE

► [eq](#) ↑

```
ret = eq(obj,aObj)
```

Description:

Tests if the two objects representing the same dates

Input:

- obj : An object of class nb_month
- aObj : Another object of class nb_month

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj,format)
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- d : Day of the date. Examples:

- > 'dd' : '01', '10'
- > 'd' : '1', '0'
- > 'd(d)' : '1', '10'
- w : Week of date. Same options as for d.
- m : Month of date. Same options as for d.

- q : Quarter of date. Examples:

- > 'q' : '1', '4'
- h : Half year of date. Same options as for h.
- y : Year of date.
 - > 'yyyy' : '2016'
 - > 'yy' : '016'
 - > 'yy' : '16'
 - < 'y' : '6'

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'
- > 'english' : 'Q'
- 'Weektext' : > 'norwegian' : 'Uke'
- > 'english' : 'Week'
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d      = nb_day.today();
date = nb_date.format2string(d,'dd.mm.yyyy');
date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m      = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q      = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y      = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');
```

See also:[toString](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► freqPlus ↑

```
obj = freqPlus(obj,incr,freq)
```

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq ↑**

frequency = nb_date.fromxlsDates2freq(dates)

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **ge ↑**

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater then or equal to another object.
Which will mean that the first input is after or the same
as the second input in the calendar

Input:

- obj : An object of class nb_month
- aObj : Another object of class nb_month

Output:

- ret : 1 if obj >= aObj, 0 else

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate ↑**

```
date = getDate(obj,freq)
```

Description:

Get the date of the object at another frequency. If the frequency
is higher the first period is returned

Input:

- obj : An object of class nb_month
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an
object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(4); % Will return a nb_quarter object
```

See also:

[nb_month.convert](#)

Written by Kenneth S. Paulsen

► **getDay** ↑

```
day = getDay(obj,first)
```

Description:

Get the first or last day of the month, given as an nb_day object

Input:

- obj : An object of class nb_month
- first : 1 : First day (always 1)
0 : Last day (Either 28, 29, 30 or 31)
2 : The first day that match the dayOfWeek property.
3 : The last day that match the dayOfWeek property.

Output:

- day : An nb_day object

Examples:

```
day = obj.getDay();      % Will return the first day of the year  
day = obj.getDay(0);    % Will return the last day of the year
```

Written by Kenneth S. Paulsen

► **getDays** ↑

```
days = getDays(obj)
```

Description:

Get all the days of the given month as a cell of nb_day objects

Input:

- obj : An object of class nb_month

Output:

- days : A cell array of nb_day objects

Examples:

```
days = obj.getDays();
```

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getFreq** ↑

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

- obj : An object of class nb_month

Output:

- freq : 12

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger ↑**

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Output:

- frequency : The frequency as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getHalfYear** ↑

```
halfYear = getHalfYear(obj)
```

Description:

Get the current half year of the month, given as an nb_semiAnnual object

Input:

- obj : An object of class nb_year

Output:

- halfYear : An nb_semiAnnual object

Examples:

```
halfYear = obj.getHalfYear();
```

Written by Kenneth S. Paulsen

► **getLatestDate** ↑

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNr** ↑

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDays** ↑

numberOfDays = getNumberOfDays(obj)

Description:

Get the number of days in the given month

Input:

- obj : An object of class nb_month

Output:

- numberOfDays : The number of days, given as double

Examples:

numberOfDays = obj.getNumberOfDays();

Written by Kenneth S. Paulsen

► **getNumberOfWeeks** ↑

numberOfWeeks = getNumberOfWeeks(obj)

Description:

Get the number of weeks in the given month

Input:

- obj : An object of class nb_month

Output:

- numberOfWeeks : The number of weeks, given as double

Written by Kenneth S. Paulsen

► **getQuarter** ↑

```
quarter = getQuarter(obj,first)
```

Description:

Get the current quarter of the month, given as an nb_quarter object

Input:

- obj : An object of class nb_mont

Output:

- quarter : An nb_quarter object

Examples:

```
quarter = obj.getQuarter();
```

Written by Kenneth S. Paulsen

► **getWeek** ↑

```
week = getWeek(obj,first,dayOfWeek)
```

Description:

Get the first or last week of the month, given as a nb_week object.

Input:

```
- obj : An object of class nb_month

- first : 1 : First week
          0 : Last week

- dayOfWeek : The weekday the given week represents when
               converted to a day. (1-7 (Monday-Sunday)). Default
               is to use the dayOfWeek property.
```

Output:

```
- week : An nb_week object
```

Examples:

```
week = obj.getWeek();      % Will return the first week of the year
week = obj.getWeek(0);    % Will return the last week of the year
```

Written by Kenneth S. Paulsen

► **getWeeks** ↑

```
weeks = getWeeks(obj,dayOfWeek)
```

Description:

Get all the weeks of the given month as a cell of nb_week objects.

Input:

```
- obj : An object of class nb_month

- dayOfWeek : The weekday the given week represents when
               converted to a day. (1-7 (Monday-Sunday)). Default
               is to use the dayOfWeek property.
```

Output:

```
- weeks : A cell array of nb_week objects
```

Examples:

```
month = nb_month(1,2020);
weeks = month.getWeeks();
```

Written by Kenneth S. Paulsen

► **getYear** ↑

```
year = getYear(obj)
```

Description:

Get the year of the month, given as a nb_year object

Input:

- obj : The object itself

Output:

- year : A nb_year object

Examples:

```
month = nb_month(1,2020);
year = month.getYear();
```

Written by Kenneth S. Paulsen

► **gt** ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater then another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_month

- aObj : An object of class nb_month

Output:

- ret : 1 if obj > aObj, 0 else

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_month.easter](#), [nb_month.ascensionDay](#), [nb_month.pentecost](#)
[nb_month.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize ↑**

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **intersect ↑**

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isDate** ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isEqualFreq** ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod ↑**

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)

m = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isLeapYear** ↑

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_month

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = objisempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : an object of class nb_month
- aObj : another object of class nb_month

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

ret = le(obj,aObj)

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

- obj : An object of class nb_month
- aObj : An object of class nb_month

Output:

- ret : 1 if obj <= aObj, 0 else

Examples:

ret = obj <= aObj;

Written by Kenneth S. Paulsen

► **lt** ↑

ret = lt(obj,aObj)

Description:

Tests if an object is less than another object. Which will mean that the first input is before the second input in the calendar

Input:

- obj : An object of class nb_month
- aObj : An object of class nb_month

Output:

- ret : 1 if obj < aObj, 0 else

Examples:

ret = obj < aObj;

Written by Kenneth S. Paulsen

► **max** ↑

d = max(obj)
[d,ind] = max(obj)
d = max(obj,aObj)

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_month objects.

Two inputs:

- obj : A scalar nb_month object.

- aObj : A scalar nb_month object.

Output:

- d : A scalar nb_month object.

- ind : The index of the last date.

Written by Kenneth Sæterhagen Paulsen

► **min ↑**

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_month objects.

Two inputs:

- obj : A scalar nb_month object.

- aObj : A scalar nb_month object.

Output:

- d : A scalar nb_month object.

- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► **ne ↑**

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_month
- aObj : An object of class nb_month

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_month.easter](#), [nb_month.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

```
out = plus(obj,aObj)
```

Description:

Makes it possible to add a given number of months to an nb_month object and receive an nb_month object which represents the month the given numbers of months ahead

Input:

- obj : A nx1 double or a nx1 nb_month object
- aObj : A nx1 double or a nx1 nb_month object

Output:

- out : A nx1 nb_year object the given number of years ahead

Examples:

```
obj = obj + 1;
```

same as

```
obj = 1 + obj;
```

Written by Kenneth S. Paulsen

► **setDayOfWeek** ↑

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_month.dayOfWeek](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **sort** ↑

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toDate** ↑

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'

> Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'

> Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'

> Quarterly : 'dd.mm.yyyy' and 'yyyyQq'

> Semiannually : 'dd.mm.yyyy' and 'yyyySs'

```

> Yearly      : 'dd.mm.yyyy' and 'yyyy'
> Secondly    : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss'
                (only supported if this version of the toolbox
                 includes the nb_second class)

- frequency :

> Daily       : 365
> Weekly      : 52
> Monthly     : 12
> Quarterly   : 4
> Semiannually : 2
> Yearly      : 1
> Secondly    : 31536000

```

Output:

```

- dObj      :

> Daily      : An nb_day object
> Weekly     : An nb_week object
> Monthly    : An nb_month object
> Quarterly  : An nb_quarter object
> Semiannually : An nb_semiAnnual object
> Yearly     : An nb_year object
> Secondly   : An nb_second object

```

Examples:

```

dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);

```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates ↑**

```
varargout = toDates(obj,periods,format,newFreq,first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : An object of class nb_month
 - periods : The periods wanted, must be given as a double array.
E.g. 1:20.
 - format : Set the format of the cellstr of dates:
 - > 'nb_date' : The output will be a vector of nb_date objects.
 - > 'default' : 'yyyyMm(m)'
 - > 'fame' : 'yyyyMm(m)'
 - > 'xls' : 'dd.mm.yyyy'
 - > 'vintage' : 'yyyymmdd'
- Formats which is dependent on the 'newFreq' input
- > 'NBNorsk' ('NBNorwegian'):
 - newFreq:
 - > 1 : 'YYYY'
 - > 2 : 'mmm. yy', e.g. 'jan. 08' and 'jul. 08'
 - > 4 : 'q.kv.yy'
 - > 12 : 'mmm. yy', e.g. mmm = 'jan'
 - > 52 : Not possible
 - > 365 : Not possible
 - (Date strings are only given for the days which ends the periods)
 - > 'NBEnglish' ('NBEngelsk'):
 - newFreq:
 - > 1 : 'YYYY'
 - > 2 : 'mmm-yy', e.g. 'jan. 08' and 'jul. 08'
 - > 4 : 'yyyyQq'
 - > 12 : 'mmm-yy', e.g. mmm = 'jan'
 - > 52 : Not possible
 - > 365 : Not possible
 - (Date strings are only given for the days which ends the periods)
 - newFreq : > 1 : yearly
> 2 : semiannually
> 4 : quarterly
> 12 : monthly
> 52 : weekly (Not possible)
> 365 : daily (Not possible)
 - first : When converting to a lower frequency you must set if you want to use the first date or the last date as

the location of the returned date. The default is to use the first.

```
1 : first  
0 : last
```

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates. Of interest when converting to lower frequencies

Examples:

```
dates          = obj.toDates(0:20);  
dates          = obj.toDates(0:20,'xls');  
[dates,locations] = obj.toDates(0:20,'NBNorsk',1);
```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start,finish,format,language,first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string method.
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_month.format2string](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toString** ↑

```
date = toString(obj,format,extra)
```

Description:

Transform the nb_month object to a string representing the date.
It will be given in the format: 'yyyyMm(m)'.

Input:

- obj : The object itself

Optional input:

```
- format : > 'xls' : 'dd.mm.yyyy'  
> 'nbnorwegian' : 'mmm. yy'  
> 'nbengelsk' or 'nbenglish' : 'Mmm-yy'  
> 'pprnorsk' or 'mprnorwegian' : 'monthtext yyyy'  
> 'pprengelsk' or 'mprenglish' : 'Monthtext yyyy'  
> 'default' : 'yyyyMm(m)'  
> 'vintage' : 'yyyymmdd'
```

- extra :

```
> 'xls':
```

Give 1 if you want the date string to represent the
first day of the month ('01.01.yyyy'), otherwise last
day of the month will be given ('31.01.yyyy'). Only
when format is given as 'xls'

```
> 'pprnorsk', 'mprnorwegian', 'pprengelsk', 'mprenglish' :
```

Give 0 if you want the month text in lowercase only.
Default is that the month text starts with an uppercase.

Output:

- date : A string with the date on the wanted format

Examples:

```
obj = nb_month(1,2012);  
date = obj.toString(); % Will give '2012M1'  
date = obj.toString('xls'); % Will give '01.01.2012'  
date = obj.toString('xls',0); % Will give '31.01.2012'  
date = obj.toString('nbnorwegian'); % Will give 'jan. 12'  
date = obj.toString('nbengelsk'); % Will give 'Jan-12'
```

```
date = obj.toString('pprnorsk'); % Will give 'Januar 2012'  
date = obj.toString('pprnorsk',0); % Will give 'januar 2012'  
date = obj.toString('pprengelsk'); % Will give 'January 2012'
```

Written by Kenneth S. Paulsen

► **today** ↑

```
obj = nb_month.today()
```

Description:

Get the current month as a nb_month object.

Output:

obj : An object of class nb_month.

Written by Kenneth Sæterhagen Paulsen

► **union** ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **vec** ↑

out = vec(obj,endD)

Description:

A method to create a vector of dates between two dates.

Input:

obj : An nb_month object which represents the start date of the vector

end : An nb_month object or a double, representing the end date or the number of periods you want in your vector

Output:

obj : A nPeriods*1 vector of nb_month objects.

Examples:

```
f = nb_month(1,1,2010);
g = nb_month(10,1,2010);
vector = f.vec(g)
```

See also:

[nb_month](#)

Written by Tobias Ingebrigtsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |  
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

■ nb_quarter

Go to: [Properties](#) | [Methods](#)

A class representing a quarterly date.

Superclasses:

nb_date

Constructor:

```
obj = nb_quarter(quarter,year)
```

Input:

- quarter : A string on one of the following date formats:

```
> 'yyyyQq'  
> 'dd.mm.yyyy'
```

Or an integer with the quarter. (Must be combined

with the year argument)

- year : Must be an integer with the year.

Output:

- obj : An nb_quarter object

Examples:

```
obj = nb_quarter('2012Q1');
obj = nb_quarter(1,2012);
```

See also:

[nb_date](#), [nb_year](#), [nb_semiAnnual](#), [nb_month](#), [nb_day](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [baseYear](#)
- [dayOfWeek](#)
- [frequency](#)
- [leapYear](#)
- [quarter](#)
- [quarterNr](#)
- [year](#)

- **baseYear** ↑

The base year. 2000.

Inherited from superclass NB_DATE

- **dayOfWeek** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass NB_DATE

- **frequency** ↑

The frequency of the date. Must be 4.

- **leapYear** ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass NB_DATE

- **quarter** ↑

The given quarter as a double

- **quarterNr** ↑

The number of quarters since the base, which is first quarter of 2000.

- **year** ↑

The year of the given quarter

Methods :

- ascensionDay
- colon
- disp
- format2string
- ge
- getDays
- getFrequencyAsInteger
- getLatestDate
- getNr
- getWeek
- gt
- intersect
- isFirstPeriod
- isempty
- le
- min
- pentecost
- sort
- toFormat
- union
- vintage2Date
- cell2Date
- convert
- easter
- freqPlus
- getDate
- getEarliestDate
- getFrequencyAsString
- getMonth
- getNumberOfDays
- getWeeks
- holidays
- isDate
- isLeapYear
- isequal
- lt
- minus
- plus
- toDate
- toString
- unique
- christmas
- date2freq
- eq
- fromxlsDates2freq
- getDay
- getFreq
- getHalfYear
- getMonths
- getNumberOfWeeks
- getYear
- initialize
- isEqualFreq
- isPeriod
- ismember
- max
- ne
- setDayOfWeek
- toDates
- today
- vec

► **ascensionDay** ↑

```
day = ascensionDay(obj,~)
```

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_quarter.easter](#), [nb_quarter.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► cell2Date ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.
- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'  
     '2011Q3'  
     '2011Q4'  
     '2012Q1'  
     '2012Q2'  
     '2012Q3'  
     '2012Q4'  
     '2013Q1'  
     '2013Q2'}
```

```
a = nb_date.cell2Date(g,4)
```

```
a =
```

```
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'
```

```
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj, ~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_quarter.ascensionDay](#), [nb_quarter.pentecost](#), [nb_quarter.easter](#)

Written by Kenneth SÅ!terhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a, d, b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;  
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► convert ↑

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_quarter object ot the wanted frequency. The return variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_quarter
- frequency : The frequency to convert to. As a scalar.
- first : 1 : first period (default)
0 : last period

Only when converting the object to a higher frequency.

Output:

- obj : An object of subclass of the nb_date class with the frequency given by the frequency input.

Examples:

```
obj = nb_quarter(1,2012);  
obj = obj.convert(1);
```

See also:

Written by Kenneth Sæterhagen Paulsen

► date2freq ↑

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'  
semiannually : 'yyyySs' or 'dd.mm.yyyy'  
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'  
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'  
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'  
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
> otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- type : 1 if the dates input is on the xls format, otherwise 0

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp** ↑

disp(obj)

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

day = easter(obj,type)

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- obj : An object of class nb_date.
- type : 'all' or 'sunday' (default).

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

`nb_easter, nb_quarter.ascensionDay, nb_quarter.pentecost`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_DATE`

► **eq** ↑

```
ret = eq(obj, aObj)
```

Description:

Tests if the two objects represent the same date

Input:

- `obj` : An object of class `nb_quarter`
- `aObj` : Another object of class `nb_quarter`

Output:

- `ret` : 1 if equal, 0 else

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj, format)
date = nb_date.format2string(obj, format, language, first)
```

Description:

Get date at a given format.

Input:

- `obj` : An object of a subclass of `nb_date`.
- `format` : A string with the format. A combination of the following patterns:
 - **d** : Day of the date. **Examples:**

```

        > 'dd'   : '01', '10'
        > 'd'    : '1', '0'
        > 'd(d)' : '1', '10'
- w : Week of date. Same options as for d.
- m : Month of date. Same options as for d.

```

- q : Quarter of date. Examples:

```

        > 'q'   : '1', '4'
- h : Half year of date. Same options as for h.
- y : Year of date.
        > 'yyyy' : '2016'
        > 'yy'   : '016'
        > 'y'    : '16'
        < 'y'    : '6'

```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

```

- 'Monthtext'   : Month of date as full text. Starting with
                  upper case.
- 'monthtext'   : Month of date as full text. Starting with
                  upper case.

- 'Quartertext' : > 'norwegian' : '. kv.'
                  > 'english'   : 'Q'

- 'Weektext'     : > 'norwegian' : 'Uke'
                  > 'english'   : 'Week'

- language : 'norwegian' or 'english' (default). Only important for the
             'monthtext' or 'Monthtext' patterns.

- first      : Give false to return the latest period when converted to a
                  higher frequency. Default is true, i.e. first period.

```

Output:

- date : A string of the date on the wanted format.

Examples:

```

d      = nb_day.today();
date = nb_date.format2string(d,'dd.mm.yyyy');
date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m      = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q      = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');

```

```
date = nb_date.format2string(m,'q. kv. yyyy');

y      = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');
```

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **freqPlus** ↑

```
obj = freqPlus(obj,incr,freq)
```

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► ge ↑

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater than or equal to another object.
Which will mean that the first input is after or the same
as the second input in the calendar

Input:

- obj : An object of class nb_quarter
- aObj : Another object of class nb_quarter

Output:

```
- ret : 1 if obj >= aObj, 0 else
```

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate** ↑

```
date = getDate(obj,freq)
```

Description:

Get the date of the object at another frequency. If the frequency is higher the first period is returned

Input:

- obj : An object of class nb_quarter
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(2); % Will return a nb_semiAnnual object
```

See also:

[nb_quarter.convert](#)

Written by Kenneth S. Paulsen

► **getDay** ↑

```
day = getDay(obj,first)
```

Description:

Get the first or last day of the quarter, given as a nb_day object

Input:

- obj : An object of class nb_quarter
- first :
 - 1 : First day (always 1)
 - 0 : Last day (Either 28, 29, 30 or 31)
 - 2 : The first day that match the dayOfWeek property.
 - 3 : The last day that match the dayOfWeek property.

Output:

- day : An nb_day object

Examples:

```
day = obj.getDay(); % Will return the first day of the quarter  
day = obj.getDay(1); % Will return the last day of the quarter
```

Written by Kenneth S. Paulsen

► getDays ↑

```
days = getDays(obj)
```

Description:

Get all the days of the given quarter as a cell of nb_day objects

Input:

- obj : An object of class nb_quarter

Output:

- days : A cell array of nb_day objects

Examples:

```
days = obj.getDays();
```

Written by Kenneth S. Paulsen

► getEarliestDate ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getFreq ↑**

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

- obj : An object of class nb_quarter

Output:

- freq : 4

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger ↑**

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString ↑**

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Output:

- frequency : The frequency as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► getHalfYear ↑

```
halfYear = getHalfYear(obj)
```

Description:

Get the half year of the quarter, given as an nb_semiAnnual object

Input:

- obj : An object of class nb_quarter

Output:

- halfYear : An nb_semiAnnual object

Examples:

```
halfYear = obj.getHalfYear(); % Will return the half year
```

Written by Kenneth S. Paulsen

► **getLatestDate** ↑

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getMonth** ↑

```
month = getMonth(obj,first)
```

Description:

Get the first or last month of the quarter, given as a nb_month object

Input:

- obj : An object of class nb_quarter
- first : 1 : first month
 0 : last month

Output:

- month : An nb_month object

Examples:

```
month = obj.getMonth();    % Will return the first month  
month = obj.getMonth(0);  % Will return the last month
```

Written by Kenneth S. Paulsen

► **getMonths** ↑

```
months = getMonths(obj)
```

Description:

Get all the months of the given quarter as a cell array of nb_month objects

Input:

- obj : An object of class nb_quarter

Output:

- months : A cell array of nb_month objects

Examples:

```
months = obj.getMonths();
```

Written by Kenneth S. Paulsen

► **getNr** ↑

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDays** ↑

```
numberOfDaysInQuarter = getNumberOfDaysInMonth(obj)
```

Description:

Get the number of days in the given quarter

Input:

- obj : An object of the class nb_quarter

Output:

- numberOfDaysInQuarter : Returns the number of days in the current quarter

Examples:

```
numberOfDays = obj.getNumberOfDaysInMonth();
```

Written by Kenneth S. Paulsen

► **getNumberOfWeeks** ↑

```
numberOfWeeks = getNumberOfWeeks(obj)
```

Description:

Get the number of weeks in the given quarter

Input:

- obj : An object of class nb_quarter

Output:

- numberOfWeeks : The number of weeks, given as double

Written by Kenneth S. Paulsen

► **getWeek** ↑

```
week = getWeek(obj,first,dayOfWeek)
```

Description:

Get the first or last week of the quarter, given as an nb_week object

Input:

- obj : An object of class nb_quarter
- first : 1 : first week
0 : last week
- dayOfWeek : The weekday the given week represents when converted to a week. (1-7 (Monday-Sunday)). Default is to use the dayOfWeek property.

Output:

- week : An nb_week object

Examples:

```
week = obj.getWeek();      % Will return the first week of the year
week = obj.getWeek(0);    % Will return the last week of the year
```

Written by Kenneth S. Paulsen

► **getWeeks** ↑

```
weeks = getWeeks(obj,dayOfWeek)
```

Description:

Get all the weeks of the given quarter as a cell of nb_week objects

Input:

- obj : An object of class nb_quarter
- dayOfWeek : The weekday the given week represents when converted to a day. (1-7 (Monday-Sunday)). Default is to use the dayOfWeek property.

Output:

- weeks : A cell array of nb_week objects

Examples:

```
quarter = nb_quarter(1,2020);
weeks = quarter.getWeeks();
```

Written by Kenneth S. Paulsen

► getYear ↑

```
year = getYear(obj)
```

Description:

Get the year of the quarter, given as a nb_year object

Input:

- obj : An object of class nb_quarter

Output:

- year : A nb_year object

Examples:

```
quarter = nb_quarter(1,2020);
year = quarter.getYear(); % Will return the current year
```

Written by Kenneth S. Paulsen

► gt ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater than another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_quarter
- aObj : An object of class nb_quarter

Output:

- ret : 1 if obj > aObj, 0 else

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_quarter.easter](#), [nb_quarter.ascensionDay](#), [nb_quarter.pentecost](#)
[nb_quarter.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize** ↑

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► intersect ↑

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► isDate ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **isEqualFreq** ↑

ret = isEqualFreq(obj,aObj)

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod** ↑

ret = isFirstPeriod(obj,freq)

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q    = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)
```

```
m    = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isLeapYear** ↑

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_quarter

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = objisempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : An object of class nb_quarter
- aObj : Another object of class nb_quarter

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

```
- varargout = ismember(obj1,obj2)
```

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

```
ret = le(obj,aObj)
```

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

- obj : An object of class nb_quarter
- aObj : An object of class nb_quarter

Output:

- ret : 1 if obj <= aObj, 0 else

Examples:

```
ret = obj <= aObj;
```

Written by Kenneth S. Paulsen

► **lt** ↑

```
ret = lt(obj,aObj)
```

Description:

Tests if an object is less then an another object. Which
will mean that the first input is before the second input in the
calendar

Input:

- obj : An object of class nb_quarter
- aObj : An object of class nb_quarter

Output:

- ret : 1 if obj < aObj, 0 else

Examples:

ret = obj < aObj;

Written by Kenneth S. Paulsen

► **max** ↑

```
d      = max(obj)
[d,ind] = max(obj)
d      = max(obj,aObj)
```

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_quarter objects.

Two inputs:

- obj : A scalar nb_quarter object.

- aObj : A scalar nb_quarter object.

Output:

- d : A scalar nb_quarter object.

- ind : The index of the last date.

Written by Kenneth Sæterhagen Paulsen

► **min** ↑

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_quarter objects.

Two inputs:

- obj : A scalar nb_quarter object.
- aObj : A scalar nb_quarter object.

Output:

- d : A scalar nb_quarter object.
- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► minus ↑

```
output = minus(obj,aObj)
```

Description:

Makes it possible to take an nb_quarter object minus another nb_quarter object and get the number of quarters between them.

It makes it also possible to take the difference between an nb_quarter object and a string one one of the following formats: 'yyyyQq', 'dd.mm.yyyy'

And last it makes it possible to subtract the number of quarters from an nb_quarter object and receive an nb_quarter representing the quarter the given number of quarters backward in time.

Input:

- obj : A date string, an nb_quarter object or an integer.
- aObj : A date string, an nb_quarter object or an integer.

Output:

- output : The number of quarters between the given quarters. As a double

or

An nb_quarter object representing the date the given number of quarter backwards in time.

Examples:

```
output = obj - aObj;           % (both objects). Returns a double  
output = obj - '2012Q2';      % Returns a double  
output = '2012Q3' - obj;      % Returns a double  
output = obj - 1;             % Returns an object  
  
same as  
  
output = 1 - obj;             % Returns an object
```

Written by Kenneth S. Paulsen

► **ne** ↑

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_quarter
- aObj : An object of class nb_quarter

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
% ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_quarter.easter](#), [nb_quarter.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

out = plus(obj,aObj)

Description:

Makes it possible to add a given number of quarters to an nb_quarter and receive a nb_quarter object which represents the quarter the given numbers of quarters ahead

Input:

- obj : A nx1 double or a nx1 nb_quarter object
- aObj : A nx1 double or a nx1 nb_quarter object

Output:

- out : A nx1 nb_quarter object with the given number of years ahead

Examples:

obj = obj + 1;

same as

obj = 1 + obj;

Written by Kenneth S. Paulsen

► **setDayOfWeek** ↑

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_quarter.dayOfWeek](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **sort ↑**

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toDate ↑**

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'
> Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'
> Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'
> Quarterly : 'dd.mm.yyyy' and 'yyyyQq'
> Semiannually : 'dd.mm.yyyy' and 'yyyySs'
> Yearly : 'dd.mm.yyyy' and 'yyyy'
> Secondly : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss'
(only supported if this version of the toolbox includes the nb_second class)

- frequency :

> Daily : 365
> Weekly : 52
> Monthly : 12
> Quarterly : 4
> Semiannually : 2
> Yearly : 1
> Secondly : 31536000

Output:

- dObj :

> Daily : An nb_day object
> Weekly : An nb_week object
> Monthly : An nb_month object
> Quarterly : An nb_quarter object

```

> Semiannually : An nb_semiAnnual object
> Yearly      : An nb_year object
> Secondly     : An nb_second object

```

Examples:

```

dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);

```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates ↑**

```
varargout = toDates(obj,periods,format,newFreq,first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : An object of class nb_quarter
- periods : The periods wanted, must be given as a double array.
E.g. 1:20.
- format : Set the format of the cellstr of dates:
 - > 'nb_date' : The output will be a vector of nb_date objects.
 - > 'default' : 'yyyyQq'
 - > 'fame' : 'yyyyQq'
 - > 'xls' : 'dd.mm.yyyy'
 - > 'vintage' : 'yyyymmdd'

Formats which is dependent on the 'newFreq' input
> 'NBNorsk' ('NBNorwegian'):

```

newFreq:
> 1   : 'YYYY'
> 2   : 'mmmm. yy', e.g. 'jan. 08' and 'jul. 08'
> 4   : 'q.kv.yy'
> 12  : Not possible
> 52  : Not possible
> 365 : Not possible

```

```

(Date strings are only given for the days which
ends the periods)

> 'NBEnglish' ('NBEngelsk'):

newFreq:
> 1    : 'YYYY'
> 2    : 'mmm-yy', e.g. 'Jan-08' and 'Jul-08'
> 4    : 'yyyyQq'
> 12   : Not possible
> 52   : Not possible
> 365  : Not possible

(Date strings are only given for the days which
ends the periods)

- newFreq : > 1    : yearly
             > 2    : semiannually
             > 4    : quarterly
             > 12   : monthly (Not possible)
             > 52   : weekly  (Not possible)
             > 365  : daily   (Not possible)

- first   : When converting to a lower frequency you must set if
you want to use the first date or the last date as
the location of the returned date. The default is to
use the first.

1 : first
0 : last

```

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates. Of interest
when converting to lower frequencies

Examples:

```

dates      = obj.toDates(0:20);
dates      = obj.toDates(0:20,'xls');
[dates,locations] = obj.toDates(0:20,'NBNorsk',1);

```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start,finish,format,language,first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string method.
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_quarter.format2string](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► [toString ↑](#)

date = toString(obj,format,first)

Description:

Transforms the nb_quarter object to a string representing the date. It will be given in the format: 'yyyyQq'.

Input:

- obj : An object of class nb_quarter
- format : > 'xls' : 'dd.mm.yyyy'
> 'nborsk' or 'nbnorwegian' : 'q.kv.yy'
> 'nbengelsk' or 'nbenglish' : 'yyyyQq'
> 'pprnorsk' or 'mprnorwegian' : 'q.kv. yyyy'
> 'pprengelsk' or 'mprenglish' : 'yyyy Qq'
> 'default' : 'yyyyQq'
> 'vintage' : 'yyyymmdd'
- first : If you want the 'xls' type date string to begin with first or last day of the month. Will only have an effect when format is set to 'xls'. Default is 1 (first day). Give any other number to get the last day.

Output:

- date : A string with the date on the wanted format

Examples:

```
obj = nb_quarter(1,2012);
date = obj.toString();                                % Will give '2012Q1'
date = obj.toString('xls');                           % Will give '01.01.2012'
date = obj.toString('xls',0);                         % Will give '31.03.2012'
date = obj.toString('nbnorsk');                      % Will give '1.kv.12'
date = obj.toString('nbengelsk');                    % Will give '2012Q1'
date = obj.toString('pprnorsk');                     % Will give '1.kv. 2012'
date = obj.toString('pprengelsk');                   % Will give '2012 Q1'
```

Written by Kenneth S. Paulsen

► today ↑

```
obj = nb_quarter.today()
```

Description:

Get the current quarter as a nb_quarter object.

Output:

- obj : An object of class nb_quarter.

Written by Kenneth Sæterhagen Paulsen

► union ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATE

► **vec** ↑

```
out = vec(obj,endD)
```

Description:

A method to create a vector of dates between to dates.

Input:

obj : An nb_quarter object which represents the start date of the vector

end : An nb_quarter object or a double, representing the end date or the number of periods you want in your vector

Output:

obj : A nPeriods*1 vector of nb_quarter objects.

Examples:

```
f = nb_quarter(1,2010);
g = nb_quarter(4,2010);
vector = f.vec(g)
```

See also:

[nb_quarter](#)

Written by Tobias Ingebrigtsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |  
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

■ **nb_semiAnnual**

Go to: [Properties](#) | [Methods](#)

A class representing a semiannual date.

Superclasses:

nb_date

Constructor:

```
obj = nb_semiAnnual(halfYear,year)
```

Input:

- halfYear : A string on one of the following date formats:

> 'yyyySq'

> 'dd.mm.yyyy'

Or an integer with the half year. (Must be combined with the year argument)

- year : Must be an integer with the year.

Output:

- obj : An nb_year object.

Examples:

```
obj = nb_year('2012');
obj = nb_year(2012);
```

See also:

[nb_date](#), [nb_year](#), [nb_quarter](#), [nb_month](#), [nb_day](#), [nb_week](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [baseYear](#)
- [dayOfWeek](#)
- [frequency](#)
- [halfYear](#)
- [halfYearNr](#)
- [leapYear](#)
- [year](#)

- **baseYear** ↑

The base year. 2000.

Inherited from superclass NB_DATE

- **dayOfWeek** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass NB_DATE

- **frequency** ↑

The frequency of the date. Must be 1.

- **halfYear** ↑

The given half year as a double.

- **halfYearNr** ↑

The number of half years since the base.
Which is the first half year of 2000

- **leapYear** ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass NB_DATE

- **year** ↑

The given year as a double

Methods:

- ascensionDay
- cell2Date
- christmas
- colon
- convert
- date2freq
- disp
- easter
- eq
- format2string
- freqPlus
- fromxlsDates2freq
- ge
- getDate
- getDay
- getDays
- getEarliestDate
- getFreq
- getFrequencyAsInteger
- getFrequencyAsString
- getLatestDate
- getMonth
- getMonths
- getNr
- getNumberOfDays
- getNumberOfWeeks
- getQuarter
- getQuarters
- getWeek
- getWeeks
- getYear
- gt
- holidays
- initialize
- intersect
- isDate
- isEqualFreq
- isFirstPeriod
- isLeapYear
- isPeriod
- isempty
- isequal
- ismember
- le
- lt
- max
- min
- minus
- ne
- pentecost
- plus
- setDayOfWeek
- sort
- toDate
- toDates
- toFormat
- toString
- today
- union
- unique
- vec
- vintage2Date

► **ascensionDay** ↑

day = ascensionDay(obj, ~)

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_semiannual.easter](#), [nb_semiannual.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► cell2Date ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.
- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'  
     '2011Q3'  
     '2011Q4'  
     '2012Q1'  
     '2012Q2'  
     '2012Q3'  
     '2012Q4'  
     '2013Q1'  
     '2013Q2'}
```

```
a = nb_date.cell2Date(g,4)
```

```
a =
```

```
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_semiAnnual.ascensionDay](#), [nb_semiannual.pentecost](#), [nb_semiannual.easter](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a,d,b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;  
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► convert ↑

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_semiAnnual object ot the wanted frequency. The return variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_semiAnnual
- frequency : The frequency to convert to. As a scalar.
- first : 1 : first period
0 : last period

Only when converting the object to a higher frequency.

Output:

- obj : An object of subclass of the nb_date class with the frequency given by the frequency input.

Examples:

```
obj = nb_semiAnnual(1,2012);
obj = obj.convert(1);
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **date2freq ↑**

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'
semiannually : 'yyyySs' or 'dd.mm.yyyy'
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
 > otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly

```
> 365 : daily  
- type      : 1 if the dates input is on the xls format,  
               otherwise 0
```

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp** ↑

```
disp(obj)
```

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

```
day = easter(obj,type)
```

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

```
- obj : An object of class nb_date.  
- type : 'all' or 'sunday' (default).
```

Output:

```
- days : All the holidays as a vector of nb_day objects.
```

See also:

[nb_easter](#), [nb_semiAnnual.ascensionDay](#), [nb_semiannual.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **eq** ↑

```
ret = eq(obj,aObj)
```

Description:

Tests if the two objects representing the same dates

Input:

```
- obj : An object of class nb_semiAnnual  
- aObj : Another object of class nb_semiAnnual
```

Output:

```
- ret : 1 if equal, 0 else
```

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj,format)  
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- d : Day of the date. Examples:

```
> 'dd'   : '01', '10'  
> 'd'    : '1', '0'  
> 'd(d)' : '1', '10'  
- w : Week of date. Same options as for d.  
- m : Month of date. Same options as for d.
```

- q : Quarter of date. Examples:

```
> 'q'   : '1', '4'  
- h : Half year of date. Same options as for h.  
- y : Year of date.  
    > 'yyyy' : '2016'  
    > 'yyy'  : '016'  
    > 'yy'   : '16'  
    < 'y'    : '6'
```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'
 > 'english' : 'Q'
- 'Weektext' : > 'norwegian' : 'Uke'
 > 'english' : 'Week'
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d      = nb_day.today();  
date = nb_date.format2string(d,'dd.mm.yyyy');  
date = nb_date.format2string(d,'d(d).m(m).yyyy');  
date = nb_date.format2string(d,'d(d) Monthtext yyyy');  
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');
```

```

m      = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q      = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y      = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');

```

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **freqPlus** ↑

obj = freqPlus(obj,incr,freq)

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **ge ↑**

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater then or equal to another object. Which will mean that the first input is after or the same as the second input in the calendar

Input:

- obj : An object of class nb_semiAnnual
- aObj : Another object of class nb_semiAnnual

Output:

- ret : 1 if obj >= aObj, 0 else

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate ↑**

```
date = getDate(obj, freq)
```

Description:

Get the date of the object at another frequency. If the frequency is higher the first period is returned

Input:

- obj : An object of class nb_semiAnnual
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(4); % Will return a nb_quarter object
```

See also:

[nb_semiAnnual.convert](#)

► **getDay** ↑

```
day = getDay(obj,first)
```

Description:

Get the first or last day of the year, given as an nb_day object

Input:

- obj : An object of class nb_semiAnnual
- first : 1 : first day
0 : last day
2 : The first day that match the dayOfWeek property.
3 : The last day that match the dayOfWeek property.

Output:

- day : An nb_day object

Examples:

```
obj = nb_semiAnnual(1,2020);  
day = obj.getDay();      % Will return the first day of the year  
day = obj.getDay(0);    % Will return the last day of the year
```

Written by Kenneth S. Paulsen

► **getDays** ↑

```
days = getDays(obj)
```

Description:

Get all the days of the given half year as a cell of nb_day objects

Input:

- obj : An object of class nb_semiAnnual

Output:

- days : A cell array of nb_day objects

Examples:

```
days = obj.getDays();
```

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getFreq** ↑

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

- obj : An object of class nb_semiAnnual

Output:

- freq : 1

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger** ↑

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Output:

- frequency : The frequency as a string

> Daily	:	'daily'
> Weekly	:	'weekly'
> Monthly	:	'monthly'
> Quarterly	:	'quarterly'
> Semiannually	:	'semiannually'
> Yearly	:	'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getLatestDate ↑**

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getMonth** ↑

```
month = getMonth(obj,first)
```

Description:

Get the first or last month of the half year, given as a nb_month object

Input:

- obj : An object of class nb_semiAnnual
- first : 1 : first month
0 : last month

Output:

- month : An nb_month object

Examples:

```
month = obj.getMonth(); % Will return the first month
month = obj.getMonth(1); % Will return the last month
```

Written by Kenneth S. Paulsen

► **getMonths** ↑

```
months = getMonths(obj)
```

Description:

Get all the months of the given half year as a cell of nb_month objects

Input:

- obj : An object of class nb_semiAnnual

Output:

- months : A cell array of nb_month objects

Examples:

```
months = obj.getMonths();
```

Written by Kenneth S. Paulsen

► **getNr ↑**

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDays ↑**

```
numberOfDays = getNumberOfDays(obj)
```

Description:

Get the number of days in the given year

Input:

- obj : An object of class nb_semiAnnual

Output:

- numberOfDays : The number of days, given as double

Examples:

```
numberOfDays = obj.getNumberOfDays();
```

Written by Kenneth S. Paulsen

► **getNumberOfWeeks ↑**

```
numberOfWeeks = getNumberOfWeeks(obj)
```

Description:

Get the number of weeks in the given half year

Input:

- obj : An object of class nb_semiAnnual

Output:

- numberOfWeeks : The number of weeks, given as double

Written by Kenneth S. Paulsen

► **getQuarter ↑**

```
quarter = getQuarter(obj,first)
```

Description:

Get the first or last quarter of the half year, given as an nb_quarter object

Input:

- obj : An object of class nb_semiAnnual

Output:

- quarter : An nb_quarter object
- first : 1 : first quarter
0 : last quarter

Examples:

```
quarter = obj.getQuarter(); % Will return the first quarter
quarter = obj.getQuarter(1); % Will return the last quarter
```

Written by Kenneth S. Paulsen

► **getQuarters** ↑

```
quarters = getQuarters(obj)
```

Description:

Get all the quarters of the given half year as a cell of nb_quarter objects

Input:

- obj : An object of class nb_semiAnnual

Output:

```
quarters : A cell array of nb_quarter objects
```

Examples:

```
quarters = obj.getQuarters();
```

Written by Kenneth S. Paulsen

► **getWeek** ↑

```
week = getWeek(obj,first,dayOfWeek)
```

Description:

Get the first or last week of the half year, given as an nb_week object

Input:

```
- obj : An object of class nb_semiAnnual  
- first : 1 : first day  
          0 : last day  
  
- dayOfWeek : The weekday the given week represents when  
               converted to a day. (1-7 (Monday-Sunday)). Default  
               is to use the dayOfWeek property.
```

Output:

```
- week : An nb_week object
```

Examples:

```
week = obj.getWeek();      % Will return the first week of the year  
week = obj.getWeek(0);    % Will return the last week of the year
```

Written by Kenneth S. Paulsen

► **getWeeks** ↑

```
weeks = getWeeks(obj,dayOfWeek)
```

Description:

Get all the weeks of the given half year as a cell of nb_week objects

Input:

```
- obj : An object of class nb_semiAnnual  
  
- dayOfWeek : The weekday the given week represents when  
               converted to a day. (1-7 (Monday-Sunday)). Default  
               is to use the dayOfWeek property.
```

Output:

```
- weeks : A cell array of nb_week objects
```

Examples:

```
sAnnual = nb_semiAnnual(1,2020);  
weeks = sAnnual.getWeeks();
```

Written by Kenneth S. Paulsen

► **getYear** ↑

```
year = getYear(obj)
```

Description:

Get the year of the half year, given as a nb_year object

Input:

- obj : An object of class nb_semiAnnual

Output:

- year : A object of class nb_year

Examples:

```
sAnnual = nb_semiAnnual(1,2020);
year      = sAnnual.getYear();
```

Written by Kenneth S. Paulsen

► **gt** ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater than another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_semiAnnual

- aObj : An object of class nb_semiAnnual

Output:

- ret : 1 if obj > aObj, 0 else

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_semiannual.easter](#), [nb_semiAnnual.ascensionDay](#), [nb_semiannual.pentecost](#)
[nb_semiannual.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize ↑**

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **intersect ↑**

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isDate** ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isEqualFreq** ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod ↑**

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)

m = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isLeapYear** ↑

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE
```

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_semiAnnual

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = objisempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : An object of class nb_semiAnnual
- aObj : Another object of class nb_semiAnnual

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

ret = le(obj,aObj)

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

- obj : An object of class nb_semiAnnual
- aObj : An object of class nb_semiAnnual

Output:

- ret : 1 if obj <= aObj, 0 else

Examples:

ret = obj <= aObj;

Written by Kenneth S. Paulsen

► **lt** ↑

ret = lt(obj,aObj)

Description:

Tests if an object is less than another object. Which will mean that the first input is before the second input in the calendar

Input:

- obj : An object of class nb_semiAnnual
- aObj : An object of class nb_semiAnnual

Output:

- ret : 1 if obj < aObj, 0 else

Examples:

ret = obj < aObj;

Written by Kenneth S. Paulsen

► **max** ↑

d = max(obj)
[d,ind] = max(obj)
d = max(obj,aObj)

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_semiAnnual objects.

Two inputs:

- obj : A scalar nb_semiAnnual object.

- aObj : A scalar nb_semiAnnual object.

Output:

- d : A scalar nb_semiAnnual object.

- ind : The index of the last date.

Written by Kenneth Sæterhagen Paulsen

► **min ↑**

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_semiAnnual objects.

Two inputs:

- obj : A scalar nb_semiAnnual object.

- aObj : A scalar nb_semiAnnual object.

Output:

- d : A scalar nb_semiAnnual object.

- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► **minus ↑**

```
output = minus(obj,aObj)
```

Description:

Makes it possible to take an nb_semiAnnual object minus another nb_semiAnnual object and get the number of half years between them.

It makes it also possible to take the difference between an nb_semiAnnual object and a string on one of the following formats: 'yyyySs', 'dd.mm.yyyy'

And last it makes it possible to subtract the number of half years from an nb_semiAnnual object and receive an nb_semiAnnual representing the half year the given number of half years backward in time.

Input:

- obj : A date string, an nb_semiAnnual object or an integer.
- aObj : A date string, an nb_semiAnnual object or an integer.

Output:

- output : The number of years between the given half years. As a double

or

An nb_semiAnnual object representing the date the given number of half years backwards in time.

Examples:

```
output = obj - aObj;      % (both objects). Returns a double
```

```
output = obj - '2012';   % Returns a double
```

```
output = '2012' - obj;  % Returns a double
```

```
output = obj - 1;        % Returns an object
```

same as

```
output = 1 - obj;       % Returns an object
```

Written by Kenneth S. Paulsen

► ne ↑

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_semiAnnual
- aObj : An object of class nb_semiAnnual

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
% ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_semiannual.easter](#), [nb_semiAnnual.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

```
out = plus(obj,aObj)
```

Description:

Makes it possible to add a given number of half years to an nb_semiAnnual object and receive an nb_semiAnnual object which represents the half year the given numbers of half years ahead

Input:

- obj : A nx1 double or a nx1 nb_semiAnnual object
- aObj : A nx1 double or a nx1 nb_semiAnnual object

Output:

- out : A nx1 nb_semiAnnual object the given number of half years ahead

Examples:

```
obj = obj + 1;
```

same as

```
obj = 1 + obj;
```

Written by Kenneth S. Paulsen

► **setDayOfWeek ↑**

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_semiAnnual.dayOfWeek](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **sort** ↑

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toDate** ↑

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'

> Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'

> Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'

> Quarterly : 'dd.mm.yyyy' and 'yyyyQq'

> Semiannually : 'dd.mm.yyyy' and 'yyyySs'

```

> Yearly      : 'dd.mm.yyyy' and 'yyyy'
> Secondly    : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss'
                (only supported if this version of the toolbox
                 includes the nb_second class)

- frequency :

> Daily       : 365
> Weekly      : 52
> Monthly     : 12
> Quarterly   : 4
> Semiannually : 2
> Yearly      : 1
> Secondly    : 31536000

```

Output:

```

- dObj      :

> Daily      : An nb_day object
> Weekly     : An nb_week object
> Monthly    : An nb_month object
> Quarterly  : An nb_quarter object
> Semiannually : An nb_semiAnnual object
> Yearly     : An nb_year object
> Secondly   : An nb_second object

```

Examples:

```

dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);

```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates ↑**

```
varargout = toDates(obj,periods,format,newFreq,first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : The object itself
 - periods : The periods wanted, must be given as a double array.
E.g. 1:20.
 - format : Set the format of the cellstr of dates:
 - > 'nb_date' : The output will be a vector of nb_date objects.
 - > 'default' : 'yyyySs'
 - > 'fame' : 'yyyySs'
 - > 'xls' : 'dd.mm.yyyy'
 - > 'vintage' : 'yyyymmdd'
- Formats which is dependent on the 'newFreq' input
- > 'NBNorsk' ('NBNorwegian'):
 - newFreq:
 - > 1 : 'YYYY'
 - > 2 : 'mmmm. yy', e.g. 'jan. 08' and 'jul. 08'
 - > 4 : Not possible
 - > 12 : Not possible
 - > 52 : Not possible
 - > 365 : Not possible
 - (Date strings are only given for the days which ends the periods)
 - > 'NBEnglish' ('NBEngelsk'):
 - newFreq:
 - > 1 : 'YYYY'
 - > 2 : 'mmmm-yy', e.g. 'Jan-08' and 'Jul-08'
 - > 4 : Not possible
 - > 12 : Not possible
 - > 52 : Not possible
 - > 365 : Not possible
 - (Date strings are only given for the days which ends the periods)
- newFreq : > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly (Not possible)
 - > 12 : monthly (Not possible)
 - > 52 : weekly (Not possible)
 - > 365 : daily (Not possible)
- first : When converting to a lower frequency you must set if you want to use the first date or the last date as

the location of the returned date. The default is to use the first.

```
1 : first  
0 : last
```

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates. Of interest when converting to lower frequencies

Examples:

```
dates          = obj.toDates(0:20);  
dates          = obj.toDates(0:20,'xls');  
[dates,locations] = obj.toDates(0:20,'NBNorsk',1);
```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start,finish,format,language,first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string method.
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_semiannual.format2string](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **toString** ↑

```
date = toString(obj, format, extra)
```

Description:

Transform the nb_semiAnnual object to a string representing the date. It will be given in the format: 'yyyyQq'.

Input:

- obj : AN object of class nb_semiAnnual

Optional input:

- format : > 'xls' : 'dd.mm.yyyy'
> 'nborsk' or 'nbnorwegian' : 'mmm. yy'
> 'nbengelsk' or 'nbenglish' : 'Mmm. yy'
> 'pprnorsk' or 'mprnorwegian' : 'Monthtext yyyy'
> 'pprengelsk' or 'mprenglish' : 'Monthtext yyyy'
> 'default' : 'yyyySs'
> 'vintage' : 'yyyymmdd'

- extra :

> 'xls':

Give 1 if you want the date string to represent the first day of the half year ('01.01.yyyy'), otherwise last day of the half year will be given ('31.06.yyyy'). Only when format is given as 'xls'

> 'pprnorsk', 'mprnorwegian', 'pprengelsk', 'mprenglish' :

Give 0 if you want the month text in lowercase only.
Default is that the month text starts with an uppercase.

Output:

- date : A string with the date on the wanted format

Written by Kenneth S. Paulsen

► **today** ↑

```
obj = nb_semiAnnual.today()
```

Description:

Get the current half year as a nb_semiAnnual object.

Output:

obj : An object of class nb_semiAnnual.

Written by Kenneth SÃ¶terhagen Paulsen

► **union** ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **vec** ↑

```
out = vec(obj,endD)
```

Description:

A method to create a vector of dates between to dates.

Input:

obj : An nb_semiAnnual object which represents the start date of the vector
end : An nb_semiAnnual object or a double, representing the end date or the number of periods you want in your vector

Output:

```
obj : A nPeriods*1 vector of nb_semiAnnual objects.
```

Examples:

```
f = nb_semiAnnual(1,2010);  
g = nb_semiAnnual(2,2011);  
vector = f.vec(g)
```

See also:

[nb_semiAnnual](#)

Written by Tobias Ingebrigtsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

```
- date : A 1xNumberOfVintages vector containing nb_date object(s).
```

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f, 4)
```

OR

```
f = {'20110622', '20111019', '20120314'} |  
t = nb_date.vintage2Date(f, 4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

■ nb_week

Go to: [Properties](#) | [Methods](#)

A class representing a weekly date.

Superclasses:

nb_date

Constructor:

```
obj = nb_week(week)  
obj = nb_week(week, dayOfWeek)  
obj = nb_week(week, year)  
obj = nb_week(week, year, dayOfWeek)
```

Input:

```
- week : A string on one of the following date formats:  
        > 'yyyyWw(w)', e.g. '2012W1'  
        > 'dd.mm.yyyy', e.g. '10.01.2012', '17.01.2012'  
        Or an integer with the week. (Must be combined  
        with the year arguments)  
- year : Must be an integer with the year.  
- dayOfWeek : Should only be provided with the date format 'yyyyWw(w)'  
    or when week and year inputs are both doubles. Default  
    is 1, which is sunday.
```

Output:

```
- obj : An nb_week object.
```

Examples:

```
obj = nb_week('01.01.2001');
```

```
obj = nb_week('2001W1');
obj = nb_week('2001W1',1);
obj = nb_week(1,2001);
obj = nb_week(1,2001,1);
```

See also:

[nb_date](#), [nb_year](#), [nb_semiAnnual](#), [nb_quarter](#), [nb_month](#), [nb_day](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [baseYear](#)
- [dayOfWeek](#)
- [frequency](#)
- [leapYear](#)
- [specialYear](#)
- [week](#)
- [weekNr](#)
- [year](#)

- [baseYear](#) ↑

The base year. 2000.

Inherited from superclass NB_DATE

- [dayOfWeek](#) ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass NB_DATE

- [frequency](#) ↑

The frequency of the date. Must be 52.

- [leapYear](#) ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass NB_DATE

- [specialYear](#) ↑

Indicator. Is 1 if the year has 53 weeks, otherwise 0.

- [week](#) ↑

The given day as a double

- [weekNr](#) ↑

The number of weeks since the base year, which is the first week of 2000

- [year](#) ↑

The year of the given week

Methods:

- ascensionDay
- colon
- disp
- format2string
- ge
- getDays
- getFrequencyAsInteger
- getLatestDate
- getNumberOfDays
- gt
- intersect
- isFirstPeriod
- isempty
- le
- min
- pentecost
- sort
- toFormat
- union
- vintage2Date
- cell2Date
- convert
- easter
- freqPlus
- getDate
- getEarliestDate
- getFrequencyAsString
- getMonth
- getQuarter
- holidays
- isDate
- isLeapYear
- isequal
- lt
- minus
- plus
- toDate
- toString
- unique
- christmas
- date2freq
- eq
- fromxlsDates2freq
- getDay
- getFreq
- getHalfYear
- getNr
- getYear
- initialize
- isEqualFreq
- isPeriod
- ismember
- max
- ne
- setDayOfWeek
- toDates
- today
- vec

► ascensionDay ↑

```
day = ascensionDay(obj,~)
```

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_week.easter](#), [nb_week.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **cell2Date** ↑

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.
- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'  
     '2011Q3'  
     '2011Q4'  
     '2012Q1'  
     '2012Q2'  
     '2012Q3'  
     '2012Q4'  
     '2013Q1'  
     '2013Q2'}
```

```
a = nb_date.cell2Date(g,4)
```

```
a =
```

```
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_week.ascensionDay](#), [nb_week.pentecost](#), [nb_week.easter](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a,d,b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;  
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **convert ↑**

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_week object ot the wanted frequency. The returned variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_week
- frequency : The frequency to convert to. As a scalar.
- first : Does nothing. Just to make the code general for all frequencies.

Output:

- obj : An object of subclass of the nb_date class with the frequency given by the frequency input.

Examples:

```
obj = nb_week(1,2012);  
obj = obj.convert(1);
```

Written by Kenneth Sæterhagen Paulsen

► **date2freq ↑**

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least to following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'  
semiannually : 'yyyySs' or 'dd.mm.yyyy'  
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'  
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'  
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'  
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
 - > otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- type : 1 if the dates input is on the xls format, otherwise 0

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp ↑**

```
disp(obj)
```

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

```
day = easter(obj,type)
```

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- obj : An object of class nb_date.
- type : 'all' or 'sunday' (default).

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_week.ascensionDay](#), [nb_week.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **eq** ↑

```
ret = eq(obj,aObj)
```

Description:

Tests if the two objects represent the same date

Input:

- obj : An object of class nb_week
- aObj : Another object of class nb_week

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string ↑**

```
date = nb_date.format2string(obj,format)
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- d : Day of the date. Examples:

```
> 'dd'   : '01', '10'
> 'd'    : '1', '0'
> 'd(d)': '1', '10'
- w : Week of date. Same options as for d.
- m : Month of date. Same options as for d.
```

- q : Quarter of date. Examples:

```
> 'q'    : '1', '4'
- h : Half year of date. Same options as for h.
- y : Year of date.
    > 'yyyy' : '2016'
    > 'yyy'  : '016'
    > 'yy'   : '16'
    < 'y'    : '6'
```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'
 > 'english' : 'Q'
- 'Weektext' : > 'norwegian' : 'Uke'
 > 'english' : 'Week'
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d = nb_day.today();
date = nb_date.format2string(d,'dd.mm.yyyy');
date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q = nb_quarter.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');
```

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **freqPlus** ↑

```
obj = freqPlus(obj,incr,freq)
```

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily : 365

> Weekly : 52

```
> Monthly      : 12
> Quarterly    : 4
> Semiannually : 2
> Yearly       : 1
```

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **ge** ↑

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater than or equal to another object.
Which will mean that the first input is after or the same
as the second input in the calendar

Input:

- obj : An object of class nb_week
- aObj : Another object of class nb_week

Output:

- ret : 1 if obj >= aObj, 0 else

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate** ↑

```
date = getDate(obj,freq)
```

Description:

Get the date of the object at another frequency.

Input:

- obj : An object of class nb_day
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(4); % Will return a nb_quarter object
```

See also:

[nb_week.convert](#)

Written by Kenneth S. Paulsen

► **getDay** ↑

```
day = getDay(obj,first)
```

Description:

Get the day of the week, given as a nb_day object

Input:

- obj : An object of class nb_week
- first : 2 : Uses the dayOfWeek property.
 - 1 : first day of the week (Monday)
 - 0 : last day of the week (Sunday)

Output:

- day : An nb_day object

Examples:

```
month = obj.getDay(); % Will return the day
```

Written by Kenneth S. Paulsen

► **getDays** ↑

```
days = getDays(obj)
```

Description:

Get all the days of the given week as a cell of nb_day objects

Input:

- obj : An object of class nb_week

Output:

- days : A cell array of nb_day objects

Examples:

```
days = obj.getDays();
```

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

```
- date : A nb_date object.  
Written by Kenneth Sæterhagen Paulsen  
Inherited from superclass NB_DATE
```

► **getFreq ↑**

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

```
- obj : An object of class nb_week
```

Output:

```
- freq : 52
```

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger ↑**

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

```
- frequency : Frequency input as a string
```

```
> Daily : 'daily'  
> Weekly : 'weekly'  
> Monthly : 'monthly'  
> Quarterly : 'quarterly'  
> Semiannually : 'semiannually'  
> Yearly : 'yearly', 'annual'
```

Output:

```
- frequency : The frequency as an integer

> Daily      : 365
> Weekly     : 52
> Monthly    : 12
> Quarterly   : 4
> Semiannually : 2
> Yearly     : 1
```

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

```
- frequency : Frequency input as an integer

> Daily      : 365
> Weekly     : 52
> Monthly    : 12
> Quarterly   : 4
> Semiannually : 2
> Yearly     : 1
```

Output:

```
- frequency : The frequency as a string
```

```
> Daily           : 'daily'  
> Weekly          : 'weekly'  
> Monthly         : 'monthly'  
> Quarterly       : 'quarterly'  
> Semiannually    : 'semiannually'  
> Yearly          : 'yearly'
```

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getHalfYear ↑**

```
halfYear = getHalfYear(obj)
```

Description:

Get the half year of the week, given as an nb_semiAnnual object

Input:

```
- obj      : An object of class nb_week
```

Output:

```
- halfYear : An nb_semiAnnual object
```

Examples:

```
halfYear = obj.getHalfYear(); % Will return the half year
```

Written by Kenneth S. Paulsen

► **getLatestDate ↑**

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getMonth** ↑

month = getMonth(obj)

Description:

Get the month of the week, given as a nb_month object

Input:

- obj : An object of class nb_week

Output:

- month : An nb_month object

Examples:

month = obj.getMonth(); % Will return the month

Written by Kenneth S. Paulsen

► **getNr** ↑

nr = getNr(obj)

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDays** ↑

numberOfDays = getNumberOfDays(obj)

Description:

Get the number of days in the given week (always 7)

Input:

- obj : An object of class nb_week

Output:

- numberOfDays : The number of days, given as double

Examples:

numberOfDays = obj.getNumberOfDays();

Written by Kenneth S. Paulsen

► **getQuarter** ↑

quarter = getQuarter(obj)

Description:

Get the quarter of the day, given as a nb_quarter object

Input:

- obj : An object of class nb_day

Output:

- quarter : An nb_quarter object

Examples:

```
quarter = obj.getQuarter(); % Will return the current quarter
```

Written by Kenneth S. Paulsen

► **getYear** ↑

```
year = getYear(obj)
```

Description:

Get the year of the week, given as a nb_year object

Input:

- obj : An object of class nb_week

Output:

- year : A nb_year object

Examples:

```
week = nb_week(1,2020);
year = week.getYear(); % Will return the current year
```

Written by Kenneth S. Paulsen

► **gt** ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater than another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_week
- aObj : An object of class nb_week

Output:

```
- ret : 1 if obj > aObj, 0 else
```

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

```
- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or  
nb_year.
```

Output:

```
- days : All the holidays as a vector of nb_day objects.
```

See also:

[nb_week.easter](#), [nb_week.ascensionDay](#), [nb_week.pentecost](#)
[nb_week.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize** ↑

```
obj = nb_date.initialize(freq, dim1, dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **intersect** ↑

cout = nb_date.intersect(varargin)

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isDate** ↑

ret = nb_date.isDate(input,freq)

Description:

Test if a string ot nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isEqualFreq** ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod** ↑

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)
```

```
m = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isLeapYear ↑**

```
ret = isLeapYear(obj)
```

Description:

Test if the current year is a leap year

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : Logical. 1 if object is leap year, 0 else

Examples:

```
ret = obj.isLeapYear();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isPeriod ↑**

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q    = nb_quarter(1,2000)
ret = isPeriod(q,1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_day

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = obj.isempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : An object of class nb_week
- aObj : Another object of class nb_week

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);  
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

```
ret = le(obj,aObj)
```

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

- obj : An object of class nb_week
- aObj : An object of class nb_week

Output:

- ret : 1 if obj <= aObj, 0 else

Examples:

```
ret = obj <= aObj;
```

Written by Kenneth S. Paulsen

► **lt** ↑

```
ret = lt(obj,aObj)
```

Description:

Tests if an object is less then an another object. Which
will mean that the first input is before the second input in the
calendar

Input:

- obj : An object of class nb_week
- aObj : An object of class nb_week

Output:

- ret : 1 if obj < aObj, 0 else

Examples:

```
ret = obj < aObj;
```

Written by Kenneth S. Paulsen

► **max** ↑

```
d      = max(obj)
[d,ind] = max(obj)
d      = max(obj,aObj)
```

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_week objects.

Two inputs:

- obj : A scalar nb_week object.

- aObj : A scalar nb_week object.

Output:

- d : A scalar nb_week object.

- ind : The index of the last date.

Written by Kenneth Sætherhagen Paulsen

► **min** ↑

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_week objects.

Two inputs:

- obj : A scalar nb_week object.
- aObj : A scalar nb_week object.

Output:

- d : A scalar nb_week object.
- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► **minus ↑**

```
output = minus(obj,aObj)
```

Description:

Makes it possible to take an nb_week object minus another nb_week object and get the number of days between them.

It makes it also possible to take the difference between an nb_week object and a string one one of the following formats:
'yyyyWw(w)', 'dd.mm.yyyy'

And last it makes it possible to subtract the number of days from an nb_week object and receive an nb_week representing the week the given number of weeks backward in time.

Input:

- obj : A date string, an nb_week object or an integer.
- aObj : A date string, an nb_week object or an integer.

Output:

- output : The number of days between the given days. As a double

or

An nb_week object representing the date the given number of days backwards in time.

Examples:

```
output = obj - aObj; % (both objects). Returns a double
```

```
output = obj - '2012W1';           % Returns a double  
output = '2012W12' - obj;          % Returns a double  
output = obj - 1;                  % Returns an object  
  
same as  
  
output = 1 - obj;                 % Returns an object
```

Written by Kenneth S. Paulsen

► ne ↑

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_week
- aObj : An object of class nb_week

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► pentecost ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_week.easter](#), [nb_week.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

out = plus(obj,aObj)

Description:

Makes it possible to add a given number of weeks to an nb_week and receive a nb_week object which represents the week the given numbers of weeks ahead

Input:

- obj : A nx1 double or a nx1 nb_week object.
- aObj : A nx1 double or a nx1 nb_week object.

Output:

- out : A nx1 nb_week object with the given number of years ahead

Examples:

obj = obj + 1;

same as

obj = 1 + obj;

Written by Kenneth S. Paulsen

► **setDayOfWeek** ↑

obj = setDayOfWeek(obj,dayOfWeek)

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_week.dayOfWeek](#)

Written by Kenneth Sølterhagen Paulsen

Inherited from superclass NB_DATE

► **sort ↑**

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth Sølterhagen Paulsen

Inherited from superclass NB_DATE

► **toDate ↑**

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'
> Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'
> Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'
> Quarterly : 'dd.mm.yyyy' and 'yyyyQq'
> Semiannually : 'dd.mm.yyyy' and 'yyyySs'
> Yearly : 'dd.mm.yyyy' and 'yyyy'
> Secondly : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmdhhnnss'
(only supported if this version of the toolbox
includes the nb_second class)

- frequency :

> Daily : 365
> Weekly : 52
> Monthly : 12
> Quarterly : 4
> Semiannually : 2
> Yearly : 1
> Secondly : 31536000

Output:

- dObj :

> Daily : An nb_day object
> Weekly : An nb_week object
> Monthly : An nb_month object
> Quarterly : An nb_quarter object
> Semiannually : An nb_semiAnnual object
> Yearly : An nb_year object
> Secondly : An nb_second object

Examples:

```
dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates** ↑

```
varargout = toDates(obj,periods,format,newFreq,first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : An object of class nb_week
- periods : The periods wanted, must be given as a double array.
E.g. 1:20.
- format : Set the format of the cellstr of dates:
 - > 'nb_date' : The output will be a vector of nb_date objects.
 - > 'default' : 'yyyyWw(w)'
 - > 'xls' : 'dd.mm.yyyy'
 - > 'vintage' : 'yyyymmdd'

Formats which is dependent on the 'newFreq' input
> 'NBNorsk' ('NBNorwegian'):

```
newFreq;
> 1   : 'YYYY'
> 2   : 'mmm. yy', e.g. 'jan. 08' and 'jul. 08'
> 4   : 'q.kv.yy'
> 12  : 'mmm. yy', e.g. mmm = 'jan'
> 52  : 'Uke w(w)-yy'
> 365 : 'dd.mm.yyyy'
```

(Date strings are only given for the days which ends the periods)

> 'NBEnglish' ('NBEngelsk'):

```
newFreq;
> 1   : 'YYYY'
> 2   : 'mmm-yy', e.g. 'jan. 08' and 'jul. 08'
> 4   : 'yyyyQq'
```

```

> 12   : 'mmm-yy', e.g. mmm = 'jan'
> 52   : 'Week w(w)-yy'
> 365  : 'dd.mm.yyyy'

(Date strings are only given for the days which
ends the periods)

- newFreq : > 1   : yearly
             > 2   : semiannually
             > 4   : quarterly
             > 12  : monthly
             > 52  : weekly
             > 365 : daily

- first   : When converting to a lower frequency you must set if
            you want to use the first date or the last date as
            the location of the returned date. The default is to
            use the first.

    1 = first
    0 = last

```

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates. Of interest
 when converting to lower frequencies

Examples:

```

dates          = obj.toDates(0:2000);
dates          = obj.toDates(0:2000,'xls');
[dates,locations] = obj.toDates(0:2000,'NBNorsk',1);

```

Written by Kenneth S. Paulsen

► **toFormat ↑**

```
dates = toFormat(start,finish,format,language,first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string
 method.

- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_week.format2string](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toString** ↑

date = toString(obj,format,first)

Description:

Transform the nb_day object to a string representing the date.

Input:

- obj : An object of class nb_day
- format :
 - > 'xls' : 'dd.mm.yyyy'
 - > 'pprnorsk' or 'mprnorwegian' : 'Uke w(w) yyyy'
 - > 'pprengelsk' or 'mprenglish' : 'Week w(w) yyyy'
 - > 'default' (otherwise) : 'yyyyWw(w)'
 - > 'vintage' : 'yyyymmdd'
- extra :
 - > 'xls' :
Give 1 if you want the date string to represent the first day of the week, otherwise last day of the week will be given. Only when format is given as 'xls'. Default is 0 when 'xls'.
 - > 'pprnorsk', 'mprnorwegian', 'pprengelsk', 'mprenglish' :
Give 0 if you want the week text in lowercase only.
Default is 1, i.e. that the week text starts with an uppercase.

Output:

- date : A string with the date on the wanted format

Examples:

```
obj = nb_week(1,2012);
date = obj.toString();                      % Will give '2012W1'
date = obj.toString('xls');                  % Will give '08.01.2012'
date = obj.toString('xls',1);                % Will give '02.01.2012'
date = obj.toString('nborsk');               % Will give '2012W1'
date = obj.toString('nbengelsk');            % Will give '2012W1'
date = obj.toString('pprnorsk');             % Will give 'Uke 1 2012'
date = obj.toString('pprnorsk',0);           % Will give 'uke 1 2012'
date = obj.toString('pprengelsk');            % Will give 'Week 1 2012'
date = obj.toString('pprengelsk',0);          % Will give 'week 1 2012'
```

Written by Kenneth S. Paulsen

► today ↑

```
obj = nb_week.today()
obj = nb_week.today(dayOfWeek)
```

Description:

Get the current week as a nb_week object.

Input:

- dayOfWeek : The weekday the given week represents when converted to a day. (1-7 (Monday-Sunday)). Default is 7 (Sunday).

Output:

obj : An object of class nb_week.

Written by Kenneth Sæterhagen Paulsen

► union ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► unique ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► vec ↑

```
out = vec(obj,endD)
```

Description:

A method to create a vector of dates between two dates.

Input:

obj : An nb_week object which represents the start date of the vector

end : An nb_week object or a double, representing the end date or the number of periods you want in your vector

Output:

```
obj : A nPeriods*1 vector of nb_week objects.
```

Examples:

```
f = nb_week(1,2010);
g = nb_week(10,2010);
vector = f.vec(g)
```

See also:

[nb_week](#)

Written by Tobias Ingebrigtsen

► vintage2Date ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

■ nb_year

Go to: [Properties](#) | [Methods](#)

A class representing a yearly date.

Superclasses:

`nb_date`

Constructor:

`obj = nb_year(year)`

Input:

- `year` : Either a string or an integer representing the year. I.e. '`yyyy`', '`yyyyY01`', '`yyyyY1`' or `yyyy`.

Output:

- `obj` : An `nb_year` object.

Examples:

```
obj = nb_year('2012');
obj = nb_year(2012);
```

See also:

`nb_date`, `nb_semiAnnual`, `nb_quarter`, `nb_month`, `nb_day`

Written by Kenneth Sæterhagen Paulsen

Properties:

- `baseYear`
- `dayOfWeek`
- `frequency`
- `leapYear`
- `year`
- `yearNr`

- **`baseYear`** ↑

The base year. 2000.

Inherited from superclass `NB_DATE`

- **`dayOfWeek`** ↑

The day which the weekly date represents when converted to or from a weekly frequency. Default is 1, which is sunday.

Inherited from superclass `NB_DATE`

- **`frequency`** ↑

The frequency of the date. Must be 1.

- **`leapYear`** ↑

Indicator for leap year. 1 if leap year.

Inherited from superclass `NB_DATE`

- **year** ↑

The given year as a double

- **yearNr** ↑

The number of years since the base year. Which is
2000

Methods:

- ascensionDay
- colon
- disp
- format2string
- ge
- getDays
- getFrequencyAsInteger
- getHalfYears
- getMonths
- getNumberOfWeeks
- getWeek
- holidays
- isDate
- isPeriod
- ismember
- max
- ne
- setDayOfWeek
- toDates
- today
- vec
- cell2Date
- convert
- easter
- freqPlus
- getDate
- getEarliestDate
- getFrequencyAsString
- getLatestDate
- getNr
- getQuarter
- getWeeks
- initialize
- isEqualFreq
- isempty
- le
- min
- pentecost
- sort
- toFormat
- union
- vintage2Date
- christmas
- date2freq
- eq
- fromxlsDates2freq
- getDay
- getFreq
- getHalfYear
- getMonth
- getNumberOfDays
- getQuarters
- gt
- intersect
- isFirstPeriod
- isequal
- lt
- minus
- plus
- toDate
- toString
- unique

► **ascensionDay** ↑

day = ascensionDay(obj, ~)

Description:

Give the ascension day of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_year.easter](#), [nb_year.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **cell2Date ↑**

```
obj = nb_date.cell2Date(cellstr,freq)
```

Description:

Converts a cellstring of dates into a nb_date object.

Input:

- cellstr : The cellstring of dates that you want to convert.
- freq : The frequency of your data. You can choose between 1, 2, 4, 12, 52, and 365.

Output:

- date : A NumberOfDatesx1 vector containing nb_date objects.

Examples:

```
g = {'2011Q2'
      '2011Q3'
      '2011Q4'
      '2012Q1'
      '2012Q2'
      '2012Q3'
      '2012Q4'
      '2013Q1'
      '2013Q2' }

a = nb_date.cell2Date(g,4)
```

```
a =
```

```
'2011Q2'  
'2011Q3'  
'2011Q4'  
'2012Q1'  
'2012Q2'  
'2012Q3'  
'2012Q4'  
'2013Q1'  
'2013Q2'
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **christmas** ↑

```
day = christmas(obj,~)
```

Description:

Give the holidays of christmas (25th and 26th) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_easter](#), [nb_year.ascensionDay](#), [nb_year.pentecost](#), [nb_year.easter](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **colon** ↑

```
cellStr = colon(a,d,b)
```

Description:

Overload the : operator. The result is a cellstr array of all the dates between the a and b (Including the a and b dates).

Input:

- a : An object of class nb_date or of a subclass of the nb_date class
- d : Increments between the dates
- b : An object of class nb_date or of a subclass of the nb_date class

Output:

- cellStr : A cellstr array of the periods between the dates represented by a and b. (Including the dates a and b). If a and b are nb_date object an empty cell is returned.

Examples:

```
dates = a:b;
dates = a:2:b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **convert ↑**

```
obj = convert(obj,frequency,first)
```

Description:

Convert the nb_year object ot the wanted frequency. The return variable will be an object which is of an subclass of the nb_date class.

Input:

- obj : An object of class nb_year
- frequency : The frequency to convert to. As a scalar.
- first : 1 : first period
0 : last period

Only when converting the object to a higher frequency.

Output:

- obj : An object of subclass of the nb_date class with the frequency given by the frequency input.

Examples:

```
obj = nb_year(2012);
obj = obj.convert(1);
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **date2freq** ↑

```
[startDate,frequency,type] = nb_date.date2freq(dates,xls)
```

Description:

A static method of the nb_date class.

Find out the frequency of the data (also from a xls(x) worksheet)

When trying to figure out the frequency of excel dates, this function must have at least two following dates

```
yearly      : 'yyyy' or 'dd.mm.yyyy'
semiannually : 'yyyySs' or 'dd.mm.yyyy'
quarterly   : 'yyyyQq' or 'yyyyKk', 'dd.mm.yyyy'
monthly     : 'yyyyMm(m)' or 'dd.mm.yyyy'
weekly       : 'yyyyWw(w)' or 'dd.mm.yyyy'
daily        : 'yyyyMm(m)Dd(d)', 'ddmonyyyy' or 'dd.mm.yyyy'
```

Input:

- dates : A cellstr of the dates you want to test the frequency of.
- xls : > 'xls' : If you want to test if the dates is on the format 'dd.mm.yyyy' also
 - > otherwise will not (default)

Output:

- startDate : An object which is a subclass of the class nb_date, with the start date of the input dates. (Either nb_quarter, nb_month, nb_semiAnnual, nb_year, nb_week or nb_day)
- frequency : The frequency of the dates given.
 - > 1 : yearly
 - > 2 : semiannually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- type : 1 if the dates input is on the xls format, otherwise 0

Examples:

```
dates = {'01.01.2012','02.01.2012','03.01.2012'};  
[startDate,frequency,type] = nb_date.date2freq(dates,'xls');
```

The output will in this example be:

- startDate : An nb_day object representing the date '01.01.2012'
- frequency : 365
- type : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **disp** ↑

```
disp(obj)
```

Description:

Sets how the nb_date object is displayed, and all subclasses of the nb_date class

Input:

- obj : An object of class nb_date or of a subclass of the nb_date class.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **easter** ↑

```
day = easter(obj,type)
```

Description:

Give the holidays or the sunday of easter of the year the date is located.

Input:

- obj : An object of class nb_date.
- type : 'all' or 'sunday' (default).

Output:

```
- days : All the holidays as a vector of nb_day objects.
```

See also:

[nb_easter](#), [nb_year.ascensionDay](#), [nb_year.pentecost](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **eq** ↑

```
ret = eq(obj,aObj)
```

Description:

Tests if the two objects representing the same dates

Input:

- obj : An object of class nb_year
- aObj : Another object of class nb_year

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj == aObj;
```

Written by Kenneth S. Paulsen

► **format2string** ↑

```
date = nb_date.format2string(obj,format)
date = nb_date.format2string(obj,format,language,first)
```

Description:

Get date at a given format.

Input:

- obj : An object of a subclass of nb_date.
- format : A string with the format. A combination of the following patterns:

- **d** : Day of the date. Examples:


```
> 'dd'   : '01', '10'
> 'd'    : '1', '0'
> 'd(d)': '1', '10'
```
- **w** : Week of date. Same options as for d.
- **m** : Month of date. Same options as for d.

- **q** : Quarter of date. Examples:

```
> 'q'   : '1', '4'
- h : Half year of date. Same options as for h.
- y : Year of date.
> 'yyyy' : '2016'
> 'yy'   : '016'
> 'yy'   : '16'
< 'y'   : '6'
```

Caution: To use the actual letter of the patterns use \ in front of the letter. E.g. '\y'.

Extra (see also the language option):

- 'Monthtext' : Month of date as full text. Starting with upper case.
- 'monthtext' : Month of date as full text. Starting with upper case.
- 'Quartertext' : > 'norwegian' : '. kv.'


```
> 'english' : 'Q'
```
- 'Weektext' : > 'norwegian' : 'Uke'


```
> 'english' : 'Week'
```
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- date : A string of the date on the wanted format.

Examples:

```
d      = nb_day.today();
date = nb_date.format2string(d,'dd.mm.yyyy');
date = nb_date.format2string(d,'d(d).m(m).yyyy');
date = nb_date.format2string(d,'d(d) Monthtext yyyy');
date = nb_date.format2string(d,'d(d). monthtext yyyy','norwegian');

m      = nb_month.today();
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'Monthtext yyyy');
date = nb_date.format2string(m,'Monthtext yyyy','norwegian');

q      = nb_quarter.today();
```

```
date = nb_date.format2string(m,'dd.mm.yyyy');
date = nb_date.format2string(m,'qQ yyyy');
date = nb_date.format2string(m,'q. kv. yyyy');

y     = nb_year.today();
date = nb_date.format2string(y,'dd.mm.yyyy');
date = nb_date.format2string(y,'yyyy');
date = nb_date.format2string(y,'\year yyyy');
```

See also:

[toString](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **freqPlus** ↑

```
obj = freqPlus(obj,incr,freq)
```

Description:

Plus the number of periods of another (lower) frequency.

E.g. dayNextYear = freqPlus(day,1,1);

Input:

- obj : An object of class nb_date.
- periods : A scalar integer with the periods to add.
- freq : The frequency of the periods argument.

Output:

- ob : An object of class nb_date.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **fromxlsDates2freq** ↑

```
frequency = nb_date.fromxlsDates2freq(dates)
```

Description:

A static method of the nb_date class.

Try to find out the date frequency of the excel dates. I.e. dates on the format 'dd.mm.yyyy'.

Input:

- Dates : A cellstr with at least two dates on the date format 'dd.mm.yyyy'.

Output:

- frequency : The frequency of the input dates given as an integer.

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Examples:

```
dates      = {'01.01.2012','02.01.2012','03.01.2012'};  
frequency = nb_date.fromxlsDates2freq(dates)
```

The output will in this example be:

- frequency : 1

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **ge ↑**

```
ret = ge(obj,aObj)
```

Description:

Tests if an object is greater then or equal to another object.
Which will mean that the first input is after or the same
as the second input in the calendar

Input:

- obj : An object of class nb_year
- aObj : Another object of class nb_year

Output:

- ret : 1 if obj >= aObj, 0 else

Examples:

```
ret = obj >= aObj;
```

Written by Kenneth S. Paulsen

► **getDate ↑**

```
date = getDate(obj, freq)
```

Description:

Get the date of the object at another frequency. If the frequency is higher the first period is returned

Input:

- obj : An object of class nb_year
- freq : The frequency you want to convert to.
 - 1 : nb_year
 - 2 : nb_semiAnnual
 - 4 : nb_quarter
 - 12 : nb_month
 - 52 : nb_week
 - 365 : nb_day

Output:

- date : The date of the object at the new frequency. As an object which is a subclass of the nb_date class.

Examples:

```
date = obj.getDate(4); % Will return a nb_quarter object
```

See also:

[nb_year.convert](#)

Written by Kenneth S. Paulsen

► **getDay** ↑

```
day = getDay(obj,first)
```

Description:

Get the first or last day of the year, given as an nb_day object

Input:

- obj : An object of class nb_year
- first : 1 : The first day
0 : The last day
2 : The first day that match the dayOfWeek property.
3 : The last day that match the dayOfWeek property.
4 : The first business day.

Output:

- day : An nb_day object

Examples:

```
obj = nb_year(2020);  
day = obj.getDay();      % Will return the first day of the year  
day = obj.getDay(0);    % Will return the last day of the year  
day = obj.getDay(2);    % Will return the first sunday of the year  
day = obj.getDay(3);    % Will return the last sunday of the year
```

Written by Kenneth S. Paulsen

► **getDays** ↑

```
days = getDays(obj)
```

Description:

Get all the days of the given year as a cell of nb_day objects

Input:

- obj : An object of class nb_year

Output:

- days : A cell array of nb_day objects

Examples:

```
days = obj.getDays();
```

Written by Kenneth S. Paulsen

► **getEarliestDate** ↑

```
date = nb_date.getEarliestDate(c)
```

Description:

Get the earliest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getFreq** ↑

```
freq = getFreq(obj)
```

Description:

Gets the frequency of the object.

Input:

- obj : An object of class nb_year

Output:

- freq : 1

Examples:

```
freq = obj.getFreq();
```

Written by Kenneth S. Paulsen

► **getFrequencyAsInteger** ↑

```
frequency = nb_date.getFrequencyAsInteger(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as an integer given a frequency as a string

Input:

- frequency : Frequency input as a string
 - > Daily : 'daily'
 - > Weekly : 'weekly'
 - > Monthly : 'monthly'
 - > Quarterly : 'quarterly'
 - > Semiannually : 'semiannually'
 - > Yearly : 'yearly', 'annual'

Output:

- frequency : The frequency as an integer
 - > Daily : 365
 - > Weekly : 52
 - > Monthly : 12
 - > Quarterly : 4
 - > Semiannually : 2
 - > Yearly : 1

Examples:

```
frequency = nb_date.getFrequencyAsInteger('yearly')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getFrequencyAsString** ↑

```
frequency = nb_date.getFrequencyAsString(frequency)
```

Description:

A static method of the nb_date class.

Get the frequency as a string given a frequency as an integer

Input:

- frequency : Frequency input as an integer

> Daily	:	365
> Weekly	:	52
> Monthly	:	12
> Quarterly	:	4
> Semiannually	:	2
> Yearly	:	1

Output:

- frequency : The frequency as a string

> Daily	:	'daily'
> Weekly	:	'weekly'
> Monthly	:	'monthly'
> Quarterly	:	'quarterly'
> Semiannually	:	'semiannually'
> Yearly	:	'yearly'

Examples:

```
frequency = nb_date.getFrequencyAsString(1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **getHalfYear** ↑

```
halfYear = getHalfYear(obj,first)
```

Description:

Get the first or last half year of the year, given as an nb_semiAnnual object

Input:

- obj : An object of class nb_year
- first : 1 : first half year
0 : last half year

Output:

- halfYear : An nb_semiAnnual object

Examples:

```
halfYear = obj.getHalfYear(); % Will return first half year
halfYear = obj.getHalfYear(1); % Will return last half year
```

Written by Kenneth S. Paulsen

► **getHalfYears** ↑

```
halfYears = getHalfYears(obj)
```

Description:

Get all the half years of the given year as a cell of nb_semiAnnual objects

Input:

- obj : An object of class nb_year

Output:

- halfYears : A cell array of nb_semiAnnual objects

Examples:

```
halfYears = obj.getHalfYears();
```

Written by Kenneth S. Paulsen

► **getLatestDate** ↑

```
date = nb_date.getLatestDate(c)
```

Description:

Get the latest date among the dates in c.

Input:

- c : A cellstr of dates.

Output:

- date : A nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getMonth** ↑

```
month = getMonth(obj,first)
```

Description:

Get the first or last month of the year, given as a nb_month object

Input:

- obj : An object of class nb_year
- first : 1 : first month
0 : last month

Output:

- month : An nb_month object

Examples:

```
month = obj.getMonth(); % Will return the first month
month = obj.getMonth(1); % Will return the last month
```

Written by Kenneth S. Paulsen

► **getMonths** ↑

```
months = getMonths(obj)
```

Description:

Get all the months of the given year as a cell of nb_month objects

Input:

- obj : An object of class nb_year

Output:

- months : A cell array of nb_month objects

Examples:

```
months = obj.getMonths();
```

Written by Kenneth S. Paulsen

► **getNr** ↑

```
nr = getNr(obj)
```

Description:

Get date number of a nb_date vector.

Input:

- obj : A vector of nb_date objects.

Output:

- nr : A double with the date numbers of each object in the nb_date vector.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **getNumberOfDays** ↑

```
numberOfDays = getNumberOfDays(obj)
```

Description:

Get the number of days in the given year

Input:

- obj : An object of class nb_year

Output:

- numberOfDays : The number of days, given as double

Examples:

```
numberOfDays = obj.getNumberOfDays();
```

Written by Kenneth S. Paulsen

► **getNumberOfWeeks ↑**

```
numberOfWeeks = getNumberOfWeeks(obj)
```

Description:

Get the number of weeks in the given year

Input:

- obj : An object of class nb_year

Output:

- numberOfWeeks : The number of weeks, given as double

Written by Kenneth S. Paulsen

► **getQuarter ↑**

```
quarter = getQuarter(obj,first)
```

Description:

Get the first or last quarter of the year, given as an nb_quarter object

Input:

- obj : An object of class nb_year

Output:

- quarter : An nb_quarter object
- first : 1 : first quarter
0 : last quarter

Examples:

```
quarter = obj.getQuarter(); % Will return the first quarter
quarter = obj.getQuarter(1); % Will return the last quarter
```

Written by Kenneth S. Paulsen

► getQuarters ↑

```
quarters = getQuarters(obj)
```

Description:

Get all the quarters of the given year as a cell of nb_quarter objects

Input:

- obj : An object of class nb_year

Output:

```
quarters : A cell array of nb_quarter objects
```

Examples:

```
quarters = obj.getQuarters();
```

Written by Kenneth S. Paulsen

► getWeek ↑

```
week = getWeek(obj,first,dayOfWeek)
```

Description:

Get the first or last week of the year, given as an nb_week object

Input:

```
- obj : An object of class nb_year  
  
- first : 1 : first day  
          0 : last day  
  
- dayOfWeek : The weekday the given week represents when  
               converted to a day. (1-7 (Monday-Sunday)). Default  
               is to use the dayOfWeek property.
```

Output:

```
- week : An nb_week object
```

Examples:

```
week = obj.getWeek();      % Will return the first week of the year  
week = obj.getWeek(0);    % Will return the last week of the year
```

Written by Kenneth S. Paulsen

► **getWeeks** ↑

```
weeks = getWeeks(obj,dayOfWeek)
```

Description:

Get all the weeks of the given year as a cell of nb_week objects

Input:

```
- obj : An object of class nb_year  
  
- dayOfWeek : The weekday the given week represents when  
               converted to a day. (1-7 (Monday-Sunday)). Default  
               is to use the dayOfWeek property.
```

Output:

```
- weeks : A cell array of nb_week objects
```

Examples:

```
year = nb_year(2020);  
weeks = year.getWeeks();
```

Written by Kenneth S. Paulsen

► **gt** ↑

```
ret = gt(obj,aObj)
```

Description:

Tests if an object is greater than another object. Which will mean that the first input is after the second input in the calendar

Input:

- obj : An object of class nb_year
- aObj : An object of class nb_year

Output:

- ret : 1 if obj > aObj, 0 else

Examples:

```
ret = obj > aObj;
```

Written by Kenneth S. Paulsen

► **holidays** ↑

```
days = holidays(obj)
```

Description:

Find the holidays of the given date object.

Input:

- obj : An object of class nb_month, nb_quarter, nb_semiAnnual or nb_year.

Output:

- days : All the holidays as a vector of nb_day objects.

See also:

[nb_year.easter](#), [nb_year.ascensionDay](#), [nb_year.pentecost](#)
[nb_year.christmas](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **initialize** ↑

```
obj = nb_date.initialize(freq,dim1,dim2)
```

Description:

Initialize a vector of date objects.

Input:

- freq : The wanted frequency of the date of today. 1,2,4,12,52 or 365.
- dim1 : The size of the first dimension.
- dim2 : The size of the second dimension.

Output:

obj : A vector of nb_date objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **intersect** ↑

```
cout = nb_date.intersect(varargin)
```

Description:

Get the intersection of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isDate** ↑

```
ret = nb_date.isDate(input,freq)
```

Description:

Test if a string or nb_date object is a date. It is also possible to restrict the test to a specific frequency.

Input:

- input : Any object
- freq : 1, 2, 4, 12, 52 or 365. Can also be empty, but then it is slower.
- locVar : A nb_struct/struct with the supported local variables. If the input is a string using the syntax '%#name', this input will be looked up and tested.

Output:

- ret : Either true or false. true if input is a date string or nb_date object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► isEqualFreq ↑

```
ret = isEqualFreq(obj,aObj)
```

Description:

Test if two date objects have the same frequency

Input:

- obj : An object which is of class nb_date or a subclass of the nb_date class
- aObj : An object which is of class nb_date or a subclass of the nb_date class

Output:

- ret : 1 if equal frequency, 0 else

Examples:

```
d = nb_day(1,1,2012);
q = nb_quarter(4,2012);
ret = d.isEqualFreq(q); % Will return 0
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **isFirstPeriod** ↑

```
ret = isFirstPeriod(obj,freq)
```

Description:

Return if the obj represent the first period a lower frequency.

Input:

- obj : An object which of a subclass of the nb_date class.
- freq : The lower frequency.

Output:

- ret : True if the object represent the first period of the lower frequency.

Examples:

```
q    = nb_quarter(1,2000)
ret = isFirstPeriod(q,1)
```

```
m    = nb_month(10,2000)
ret = isFirstPeriod(m,4)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isPeriod** ↑

```
ret = isPeriod(obj,period)
```

Description:

Return true if the obj represent the given period, i.e. if you want to test if a nb_quarter object represent the first period of any year you can use isPeriod(obj,1).

Input:

- obj : An object which of a subclass of the nb_date class.
- period : The period to test for.

Output:

- ret : True if the object represent the period to test for, else false.

Examples:

```
q = nb_quarter(1,2000)
ret = isPeriod(q,1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if the object is empty

Input:

- obj : An object of class nb_year

Output:

- ret : 1 if empty, 0 else. Always false for vectors of objects.

Examples:

```
ret = obj.isempty();
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isequal** ↑

```
ret = isequal(obj,aObj)
```

Description:

Tests if the two objects representing the same date

Input:

- obj : An object of class nb_year
- aObj : Another object of class nb_year

Output:

- ret : 1 if equal, 0 else

Examples:

```
ret = obj.isequal(aObj);  
ret = isequal(obj,aObj);
```

Written by Kenneth S. Paulsen

► **ismember** ↑

- varargout = ismember(obj1,obj2)

Description:

Utilizes the inbuilt MATLAB function 'ismember' for nb_date objects.

Input:

- obj1 : A nb_date object
- obj2 : A nb_date object

Output:

- varargout : See help on MATLAB function 'ismember'.

See also:

[ismember](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

► **le** ↑

ret = le(obj,aObj)

Description:

Tests if an object is less then or equal to another object.
Which will mean that the first input is before or the same as
the second input in the calendar.

Input:

```
- obj    : An object of class nb_year  
- aObj   : An object of class nb_year
```

Output:

```
- ret    : 1 if obj <= aObj, 0 else
```

Examples:

```
ret = obj <= aObj;
```

Written by Kenneth S. Paulsen

► **lt** ↑

```
ret = lt(obj,aObj)
```

Description:

Tests if an object is less than another object. Which will mean that the first input is before the second input in the calendar

Input:

```
- obj    : An object of class nb_year  
- aObj   : An object of class nb_year
```

Output:

```
- ret    : 1 if obj < aObj, 0 else
```

Examples:

```
ret = obj < aObj;
```

Written by Kenneth S. Paulsen

► **max** ↑

```
d      = max(obj)  
[d,ind] = max(obj)  
d      = max(obj,aObj)
```

Description:

Find the max date.

Input:

One input:

- obj : A vector of nb_year objects.

Two inputs:

- obj : A scalar nb_year object.
- aObj : A scalar nb_year object.

Output:

- d : A scalar nb_year object.
- ind : The index of the last date.

Written by Kenneth Sæterhagen Paulsen

► **min ↑**

```
d      = min(obj)
[d,ind] = min(obj)
d      = min(obj,aObj)
```

Description:

Find the min date.

Input:

One input:

- obj : A vector of nb_year objects.

Two inputs:

- obj : A scalar nb_year object.
- aObj : A scalar nb_year object.

Output:

- d : A scalar nb_year object.
- ind : The index of the first date.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

```
output = minus(obj,aObj)
```

Description:

Makes it possible to take an nb_year object minus another nb_year object and get the number of years between them.

It makes it also possible to take the difference between an nb_year object and a string on one of the following formats:
'yyyy', 'dd.mm.yyyy'

And last it makes it possible to subtract the number of years from an nb_year object and receive an nb_year representing the year the given number of years backward in time.

Input:

- obj : A date string, an nb_year object or an integer.
- aObj : A date string, an nb_year object or an integer.

Output:

- output : The number of years between the given years. As a double
or

An nb_year object representing the date the given number of years backwards in time.

Examples:

```
output = obj - aObj;      % (both objects). Returns a double
output = obj - '2012';    % Returns a double
output = '2012' - obj;   % Returns a double
output = obj - 1;         % Returns an object
same as
output = 1 - obj;        % Returns an object
```

Written by Kenneth S. Paulsen

► **ne** ↑

```
ret = ne(obj,aObj)
```

Description:

Tests if the two objects not representing the same date

Input:

- obj : An object of class nb_year
- aObj : An object of class nb_year

Output:

- ret : 1 if obj is not equal to aObj, 0 else

Examples:

```
ret = obj ~= aObj;
```

Written by Kenneth S. Paulsen

► **pentecost** ↑

```
day = pentecost(obj,~)
```

Description:

Give the pentecost (monday) of the year the date is located.

Input:

- obj : An object of class nb_date.

Output:

- days : The holiday as a nb_day object.

See also:

[nb_easter](#), [nb_year.easter](#), [nb_year.ascensionDay](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **plus** ↑

```
out = plus(obj,aObj)
```

Description:

Makes it possible to add a given number of years to an nb_year object and receive an nb_year object which represents the year the given numbers of years ahead

Input:

- obj : An nx1 double or a nx1 nb_year object.
- aObj : An nx1 double or a nx1 nb_year object.

Output:

- out : A nx1 nb_year object the given number of years ahead.

Examples:

```
obj = obj + 1;
```

same as

```
obj = 1 + obj;
```

Written by Kenneth S. Paulsen

► **setDayOfWeek** ↑

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day of week number of a set of nb_date objects.

Input:

- obj : A nb_date object. May be a vector or matrix.

Output:

- obj : A nb_date object with the same size as the obj input.

See also:

[nb_year.dayOfWeek](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **sort ↑**

```
[obj,ind] = sort(obj)
```

Description:

Sort a vector of nb_date objects.

Input:

- obj : A vector of nb_date objects.

Output:

- obj : A sorted vector of nb_date objects.

- ind : The index of the sorting.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATE

► **toDate ↑**

```
dObj = nb_date.toDate(date,frequency)
```

Description:

A static method of the nb_date class.

Transform a given date string to the corresponding date object given that the frequency is already known.

If the frequency is not known, use the method date2freq() (handles also excel dates) or getFreq() (Only nb_ts dates).

Input:

- date :

Supported date formats:

> Daily : 'dd.mm.yyyy' and 'yyyyMm(m)Dd(d)'

> Weekly : 'dd.mm.yyyy' and 'yyyyWw(w)'

> Monthly : 'dd.mm.yyyy' and 'yyyyMm(m)'

```

> Quarterly      : 'dd.mm.yyyy' and 'yyyyQq'
> Semiannually   : 'dd.mm.yyyy' and 'yyyySs'
> Yearly        : 'dd.mm.yyyy' and 'yyyy'
> Secondly       : 'yyyy-mm-dd hh:nn:ss' and 'yyyymmddhhnnss'
                  (only supported if this version of the toolbox
                  includes the nb_second class)

- frequency :

> Daily          : 365
> Weekly         : 52
> Monthly        : 12
> Quarterly      : 4
> Semiannually   : 2
> Yearly         : 1
> Secondly       : 31536000

```

Output:

```

- dObj      :

> Daily      : An nb_day object
> Weekly     : An nb_week object
> Monthly    : An nb_month object
> Quarterly  : An nb_quarter object
> Semiannually : An nb_semiAnnual object
> Yearly     : An nb_year object
> Secondly   : An nb_second object

```

Examples:

```

dObj = nb_date.toDate('2012M1D1',365);
dObj = nb_date.toDate('2012M1',12);
dObj = nb_date.toDate('2012Q1',4);
dObj = nb_date.toDate('2012S1',2);
dObj = nb_date.toDate('2012',1);

```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATE

► **toDates** ↑

```
varargout = toDates(obj, periods, format, newFreq, first)
```

Description:

Get a cellstr array of the dates ahead

Input:

- obj : An object of class nb_year
- periods : The periods wanted, must be given as a double array.
E.g. 1:20.
- format : > 'xls' : 'dd.mm.yyyy'
> 'vintage' : 'yyyymmdd'
> 'nb_date' : Vector of nb_year
> otherwise : 'yyyy'
- newFreq : Give a error if you are trying to change the frequency. 1 is the only appropriate input value.
- first : Give 1 if you want the date strings to represent the first day of the year ('01.01.yyyy'), otherwise last day of the year will be given ('31.12.yyyy'). Only when format is given as 'xls'

Output:

- varargout{1} : A cell array of the wanted dates.
- varargout{2} : Locations of the returned dates.

Examples:

```
dates      = obj.toDates(0:20);
dates      = obj.toDates(0:20,'xls');
```

Written by Kenneth S. Paulsen

► **toFormat** ↑

```
dates = toFormat(start, finish, format, language, first)
```

Description:

Convert dates vector to a given format.

Input:

- start : An object of a subclass of the nb_date class.
- finish : An object of a subclass of the nb_date class.
- format : See the format input to the nb_date.format2string method.
- language : 'norwegian' or 'english' (default). Only important for the 'monthtext' or 'Monthtext' patterns.
- first : Give false to return the latest period when converted to a higher frequency. Default is true, i.e. first period.

Output:

- vint : A cellstr.

See also:

[nb_year.format2string](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **toString** ↑

date = toString(obj,format,first)

Description:

Transform the nb_year object to a string representing the date.

Input:

- obj : An object of class nb_year
- format : > 'xls' : 'dd.mm.yyyy'
 > 'vintage' : 'yyyymmdd'
 > otherwise : 'yyyy'
- first : Give 1 if you want the date string to represent the first day of the year ('01.01.yyyy'), otherwise last day of the year will be given ('31.12.yyyy'). Only when format is given as 'xls'

Output:

- date : A string on the wanted format

Examples:

```
obj = nb_year(2012);
date = obj.toString();           % Will give '2012'
date = obj.toString('xls');     % Will give '31.12.2012'
date = obj.toString('xls',1);   % Will give '01.01.2012'
```

Written by Kenneth S. Paulsen

► **today** ↑

```
obj = nb_year.today()
```

Description:

Get the current year as a nb_year object.

Output:

obj : An object of class nb_year.

Written by Kenneth Sæterhagen Paulsen

► **union** ↑

```
cout = nb_date.union(varargin)
```

Description:

Get the union of dates.

Input:

- varargin : Each input must be a cellstr of dates.

Output:

- cout : A cellstr of dates.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **unique** ↑

```
obj = unique(obj)
```

Description:

Get unique elements of a vector of nb_date objects.

Input:

- in : A vector of nb_date objects.

Output:

- out : A unique (sorted) vector of nb_date objects.

- IA : out = in(IA)

- IC : A = C(IC)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATE

► **vec** ↑

out = vec(obj, endD)

Description:

A method to create a vector of dates between two dates.

Input:

obj : An nb_year object which represents the start date of the vector

end : An nb_year object or a double, representing the end date or the number of periods you want in your vector

Output:

obj : A nPeriods*1 vector of nb_year objects.

Examples:

```
f = nb_year(2010);
g = nb_year(2014);
vector = f.vec(g)
```

See also:

[nb_year](#)

Written by Tobias Ingebrigtsen

► **vintage2Date** ↑

```
date = nb_date.vintage2Date(vintage,freq)
```

Description:

This method converts a vintage date into a nb_date object with the wanted frequency. You can also convert a cellstring containing multiple vintage dates.

Input:

- vintage : Either the vintage date that you want to convert, or a cellstring containing the vintage dates you want to convert.
- freq : The wanted frequency. Your options are 1, 4, 12 or 365.

Output:

- date : A 1xNumberOfVintages vector containing nb_date object(s).

Examples:

```
f = '20110622'  
t = nb_date.vintage2Date(f,4)
```

OR

```
f = {'20110622','20111019','20120314'} |  
t = nb_date.vintage2Date(f,4)
```

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATE

◆ **nb_dateminus**

```
diff = nb_dateminus(date1,date2)
```

Description:

This function calculates how many quarters or months there is between two dates (given as strings).

Inputs:

- date1 : Must be on one of the following date formats:

- > 'yyyyQq', e.g. '2011Q1'
- > 'yyyyMm(m)', e.g. '2011M1'
- > 'yyyy', e.g. 2011

- date2 : Must be on one of the following date formats:

- > 'yyyyQq', e.g. '2011Q1'

```
> 'yyyyMm(m)', e.g. '2011M1'  
> 'yyyy', e.g. 2011
```

Output:

- diff : Number of quarters/months/years between date1 and date2
(date1 minus date2)

Examples:

```
diff = nb_dateminus('2012Q4','2012Q1');  
diff = nb_dateminus('2012M4','2012M1');  
diff = nb_dateminus('2014','2012');
```

See also:

[nb_quarter](#), [nb_month](#), [nb_year](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_dateplus**

```
date = nb_dateplus(datein,plus)
```

Description:

You can specify a date and get the date "plus" period(s) ahead.

Input:

- datein : A date string on one of the following formats
 - > 'yyyyQq', e.g. '2011Q1'.
 - > 'yyyyMm', e.g. '2011M1'.
 - > 'yyyy', e.g. '2011'
- added : An integer with the number of added months, quarters or years ahead

Output:

date : A date string at the same date format as the input, but with the added periods.

Example of use of function:

```
dateString = dateplus('2011Q1',1);      % '2011Q2'  
dateString = dateplus('2011Q1',-1);      % '2010Q4'
```

See also:

[nb_quarter](#), [nb_month](#), [nb_year](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_easter**

easter = nb_easter(years)

Description:

A function to identify the date of easter for a given year. Utilizes Gauss' Easter Algorithm.

https://en.wikipedia.org/wiki/Computus#Gauss's_Easter_algorithm

Input:

- years : A double vector of years

Output:

- easter : A cellstring with the dates of easter corresponding to each year

Examples:

```
years = 1950:2050;
easter = nb_easter(years)
```

Written by Tobias Ingebrigtsen

◆ **nb_isDate**

[ret,date] = nb_isDate(in)

Description:

Check if any type of input is a date as a one line char or a nb_date object.

Input:

- in : Any input.

Output:

```
- ret   : true or false
- date : An object of class nb_date.
```

See also:

[nb_date](#), [nb_date.date2freq](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_sameDateFormat**

returned = nb_sameDateFormat(date1,date2)

Description

Compares two string, and test if the dates are of the same frequency. Return true if the two dates have the same frequency. The following date formats are supported by this function:

- Daily : 'yyyyMm(m)Dd(dd)'
- Weekly : 'yyyyW(w)'
- Monthly : 'yyyyMm(m)'
- Quarterly : 'yyyyQq'
- Semiannually : 'yyyySs'
- Yearly : 'yyyy'

Input:

- date1 : A string with the date.
- date2 : A string with the date.

Output:

- returned : Logical. 1 if both dates have same frequency, otherwise 0.

Examples:

returned = nb_sameDateFormat('2013Q4','2012Q1'); % true

See also:

[nb_date.date2freq](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_xlsDates2FAMEVintage**

```
d          = nb_xlsDates2FAMEVintage(dates)
[d,years,months,days] = nb_xlsDates2FAMEVintage(dates)
```

Description:

Convert dates from the format 'dd.mm.yyyy' to 'yyyymmdd'.

Input:

- dates : A cellstr or char array where each element is on the format 'dd.mm.yyyy'.

Output:

- d : A cellstr where each element is on the format 'yyyymmdd'.
- years : A double with the years of the provided dates.
- months : A double with the months of the provided dates.
- days : A double with the days of the provided dates.

Written by Kenneth Sæterhagen Paulsen

25.2 Data management classes and functions

- nb_DataCollection
- nb_bd
- nb_cell
- nb_cell2obj
- nb_checkPostFix
- nb_checkPreFix
- nb_covidDates
- nb_covidDummyNames
- nb_createDefaultLink
- nb_cs
- nb_data
- nb_deleteDefaultWorksheets
- nb_eval
- nb_excelCellOffset
- nb_excelRange
- nb_findIndex
- nb_input2String
- nb_isQorM
- nb_letter2num
- nb_lookUpVariables
- nb_math_ts
- nb_mergeSources
- nb_num2letter
- nb_readExcel
- nb_readExcelMorePages
- nb_readExcelMoreSheets
- nb_readMat
- nb_struct2nb_dataSource
- nb_ts
- nb_validpath
- nb_xlsGetSheets
- nb_xlsNum2Column
- nb_xlsread
- nb_xlswrite

■ nb_DataCollection

Go to: [Properties](#) | [Methods](#)

A class for storing nb_ts, nb_cs and/or nb_data objects

Constructor:

obj = nb_DataCollection(filename)

Input:

- filename : A string with the names of an excel spreadsheet.

If given this input will be given to the nb_readExcelMoreSheets, so see the documentation of this function for more one the needed format of the excel spreadsheet.

If this input is not given, a empty nb_DataCollection object is constructed.

Output:

- obj : An object of class nb_DataCollection

Examples:

```
obj = nb_DataCollection();
obj = nb_DataCollection('excelFileName');
```

See also:

[nb_ts](#), [nb_cs](#), [nb_readExcelMoreSheets](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [list](#)
- [objectsID](#)

- [list](#)

An object of class nb_List storing the data objects

- [objectsID](#)

A cellstr with the identifiers of the stored objects.

Methods:

- [add](#)
- [convert](#)
- [disp](#)
- [get](#)
- [getVariables](#)
- [merge](#)
- [remove](#)
- [reset](#)
- [saveData](#)

► [add](#)

```
obj = add(obj,dataObject,id)
```

Description:

Adding an object of class nb_ts or nb_cs to a nb_DataCollection object.

Caution : The added object can only have one page (dataset)

Input:

- obj : An object of class nb_DataCollection
- dataObject : An nb_ts or nb_cs object with only one page (dataset).
- id : A string with the identifier of the provided object.

Output:

- obj : An object of class nb_DataCollection with the added data object.

Examples:

```
obj      = nb_DataCollection();
dataObject = nb_ts([2,2],'', '2012Q1', 'Var1');
obj      = obj.add(dataObject,'Dataset1');
```

Written by Kenneth S. Paulsen

► convert ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert all the nb_ts object of the object. I.e. use their convert method with the same inputs as given to this method.

Input:

- obj : An object of class nb_DataCollection
- freq : The new frequency of the data. As an integer:
 - 1 : yearly
 - 4 : quarterly
 - 12 : monthly
 - 365 : daily
- method : The method to use. Either:

From a high frequency to a low:

- 'avarage' : Takes averages over the subperiods
- 'sum' : Takes sums over the subperiods
- 'discrete' : Takes last observation in each subperiod (Default)

From a low frequency to a high:

- 'linear' : linear interpolation (Default)
- 'cubic' : shape-preserving piecewise cubic interpolation
- 'spline' : piecewise cubic spline interpolation
- 'none' : no interpolation. All data in between is given by nans (missing values)

Caution: This input must be given if you provide some of the optional inputs.

Optional input:

- 'includeLast' : Give this string as input if you want to include the last period, even if it is not finished. Only when converting to lower frequencies
 - E.g: `obj = obj.convert(1,'average',...
'includeLast');`
- 'rename' : Changes the postfixes of the variables names.
 - E.g. If you covert from the frequency 12 to 4 the prefixes of the variable names changes from 'MUA' to 'QUA', if the postfix exist.
 - E.g: `obj = obj.convert('yearly','average',...
'rename');`

Optional input `obj.convert(...,'inputName','inputValue',...)`:

- 'interpolateDate' : The input after this input must either be 'start' or 'end'.
 - Date of interpolation. Where 'start' means to interpolate the start date of the periods of the old frequency, while 'end' uses the end date of the periods.
 - Default is 'start'.
 - Caution : Only when converting to higher frequency.
 - E.g: `obj = obj.convert(365,'linear',...
'interpolateDate','end');`

Output:

- obj : An object of class nb_DataCollection with all the stored nb_ts object converted to the new freqency.

Examples:

```
obj = obj.convert(4,'average');
obj = obj.convert(365,'linear','interpolateDate','end');
obj = obj.convert(1,'average','includeLast');
obj = obj.convert(1,'average','rename');
```

Written by Kenneth S. Paulsen

► **disp** ↑

```
disp(obj)
```

Description:

Overrun how the object is displayed in MATLAB (On the command line)

Input:

- obj : An object of class nb_DataCollection

Output:

The object displayed on the command line

Examples:

```
obj
```

Written by Kenneth S. Paulsen

► **get** ↑

```
obj = get(obj,id)
```

Description:

Get the object with the given identifier

Input:

- obj : An object of class nb_DataCollection
- id : The identifier of the object you want to get. As a string.

Output:

- dataObject : The data object which match the identifier.
Either a nb_cs, nb_ts or nb_data

Examples:

```
dataObject = get(obj,'Dataset1');
```

Written by Kenneth S. Paulsen

► **getVariables ↑**

```
obj = getVariables(obj,vars,interpolateDate,method,rename)
```

Description:

Get the variables given by the vars input stored in the nb_DataCollection object.

Caution : Be aware that the variables you try to get can only represent time series or cross-sectional data.

Caution : If the variables represent different frequencies the lowest frequencies will be converted to the highest frequency.

Input:

- obj : An object of class nb_DataCollection
- vars : A cellstr array of the variable you want to load.
- interpolateDate :
 - How to transform the database with the lowest frequency.
 - 'start' : The interpolated data are taken as given at the start of the periods. I.e. use 01.01.2012 and 01.01.2013 when interpolating data on yearly frequency. (Default)
 - 'end' : The interpolated data are taken as given at the end of the periods. I.e. use 31.12.2012 and 31.12.2013 when interpolating data on yearly frequency.
- method : The method to use when converting:
 - 'linear' : Linear interpolation
 - 'cubic' : Shape-preserving piecewise cubic interpolation

```

        - 'spline'    : Piecewise cubic spline
                         interpolation
        - 'none'      : No interpolation. All data in
                         between is given by nans
                         (missing values) (Default)

- rename          : > 'on'   : Renames the prefixes (default)
                     > 'off'  : Do not rename the prefixes

For more see the 'rename' option of the
convert method.

```

Output:

- obj : An object of class nb_ts or nb_cs.

Examples:

Written by Kenneth S. Paulsen

► **merge** ↑

obj = merge(obj,aObj)

Description:

Merge two objects of class nb_DataCollection

Collect each object stored in the two input objects of class nb_DataCollection and store it in one nb_DataCollection object.

Input:

- obj : An object of class nb_DataCollection
- aObj : An object of class nb_DataCollection

Output:

- obj : An object of class nb_DataCollection, where all the data objects of the two input objects of class nb_DataCollection are stored.

Examples:

obj = merge(obj,aObj)

Written by Kenneth S. Paulsen

► **remove** ↑

```
obj = remove(obj,id)
```

Description:

Remove a data object from the given object. (Using the objects identifier)

Input:

- obj : An object of class nb_DataCollection
- id : The identifier of the object you want to remove. As a string.

Output:

- obj : An object of class nb_DataCollection with the data object with the given identifier removed.

Examples:

```
obj = obj.remove('Dataset1');
```

Written by Kenneth S. Paulsen

► **reset ↑**

```
obj = reset(obj,dataObject,id)
```

Description:

Resetting an object of class nb_ts or nb_cs of an nb_DataCollection object.

Caution : The reset object can only have one page (dataset)

Input:

- obj : An object of class nb_DataCollection
- dataObject : An nb_ts or nb_cs object with only one page (dataset).
- id : A string with the identifier of the reset object.

Output:

- obj : An object of class nb_DataCollection with the reset data object.

Examples:

```
obj      = obj.reset(dataObject,'Dataset1');
```

Written by Kenneth S. Paulsen

► **saveData** ↑

```
saveData(obj,saveName,descriptionPage)
```

Description:

Write object to excel, where each stored object is saved in
seperate worksheets

Input:

- obj : An object of class nb_DataCollection
- saveName : A string with the wanted name of the saved
excel file
- descriptionPage : Give 1 if you want a description page.
Default is 0.

Output:

The data of the object is saved to excel with the given save
name. Each worksheet is saved with the names of the objects
identifiers.

Examples:

```
obj      = nb_DataCollection();  
dataObject = nb_ts([2,2],'', '2012Q1', 'Var1');  
obj      = obj.add(dataObject,'Dataset1');  
saveData(obj,'test')  
saveData(obj,'test',1)
```

Written by Kenneth S. Paulsen

■ **nb_bd**

Go to: [Properties](#) | [Methods](#)

A class for storing vintage / business-day data or simply time-series
with a large share of missing values.

Constructor:

```
obj = nb_bd(datasets, nameOfDatasets, startDate, variables)
obj = nb_bd(datasets, nameOfDatasets, startDate, variables, sorted)
```

Input:

- datasets:

Input can be one of the data types listed below:

- xls(x) : A name of the excel spreadsheet to import. (With or without extension.). As a string.

Caution : If you read a specific worksheet, and want the object to be updateable you must provide the extension.

- mat : Name(s) of the .mat file(s) to import. (With or without extension.) As a string. This .mat file must store a struct where each field stores the data of a variable. The data stored at each field must be doubles with same size. The only exception is that you must include a field called 'dates', storing a cellstr with the dates of the data. (This cellstr must have the same size as the provided data (size of the first dimension)).

Caution : The data stored as seperate field could have more pages (size of the third dimension could be lager then 1). Again All the data of the variables must be of the same size.

- double : As a double matrix. Each page of the matrix will be added as separate datasets.

- struct : As a structure of double matrices. Each field is added as a separate datasets.

You can also give a cell array consisting of one or a combination of the data types mentioned above. Each cell will be added as a new dataset.

Extra: Use the method structure2nb_bd function to add all the fields of the structure as variables in an nb_bd object. (If each field has more page then 1, then each page will be a new datasets). See the method structure2nb_bd for more on the supported format of the structure you can provide as input.

- nameOfDatasets :

Input will depend on the data type provided to the datasets input:

- xls(x) : A string with the wanted dataset name. Default is the excel file name. If the provided string is a sheet of the read spreadsheet that specific worksheet will be read.

- mat : A string with the wanted dataset name. Default is the .mat file name.

- double : A string with the wanted dataset name. Default is 'Dataset1'.

Caution: When you give double matrices which has more pages than one, either directly or through a struct. Each of the pages will be added as a dataset. The name of this datasets will, if the input nameOfDatasets has the same size as the input datasets, get the name nameOfDatasets{jj}<jj>, where jj is the page number, or else get the name 'Database<jj>'.

- struct : This input has no consequence. The dataset names will be given by the fieldnames.

- A cell array of the listed types above:

If you give a cell array to the datasets input, this input must be a cell array of strings. If you give a cell array which doesn't have the same size as the cell array given to the datasets input, the datasets which is left gets the name 'Database<jj>', where <jj> stands for this added datasets number.

If you give an empty cell array, i.e. {} all the datasets get the names; 'Database<jj>', where <jj> stands for the added datasets number. Except when the datasets input are given by a cellstr with the excel file names or .mat file names, then the default dataset names will be set to the same cellstr. Or if some elements of the cell is a struct, then the datasets provided through the struct will get their fieldnames as the dataset names.

- dates :

A cell array of the type names of the data, as strings. Only needed when you give a double or a struct of doubles as the datasets input. The number of dates must be the same as the size of the data (1- dimension).

- variables :

A cell array of the variable names of the data, as strings. Only needed when you give a double or a struct of doubles as the datasets input. The number of variables must be the same as the size of the data (2- dimension).

- sorted : true or false. Default is true.

Output:

- obj : An object of class nb_bd.

Examples:

```

- xls(x):

    One excel spreadsheet

    obj = nb_bd('test1')

    More excel spreadsheets

    obj = nb_bd({'test1','test2',...})

- mat:

    One .mat file:

    obj = nb_bd('test1')

    More .mat files:

    obj = nb_bd({'test1','test2',...})

    same as

    obj = nb_bd({'test1','test2',...},{'test1','test2',...})

- double matrix:

    One matrix

    obj = nb_bd(double1,'','2000Q1',{'Var1','Var2',...})

    obj = nb_bd(double1,'test1','2000Q1',{'Var1','Var2',...})

```

See also:

[nb_graph_bd](#), [nb_ts](#), [nb_graph_ts](#)

Written by Per Bjarne Bye

Properties:

- [data](#)
- [ignorenan](#)
- [numberOfDatasets](#)
- [userData](#)
- [dataNames](#)
- [indicator](#)
- [numberOfObservations](#)
- [variables](#)
- [endDate](#)
- [localVariables](#)
- [numberOfVariables](#)
- [frequency](#)
- [locations](#)
- [startDate](#)

- **data** ↑

The data of the object. As a double or logical.

Inherited from superclass NB_DATASOURCE

- **dataNames** ↑

The names of the different dataset. As a cellstr.

Inherited from superclass NB_DATASOURCE

- **endDate** ↑

The end date of the data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_week, nb_month, nb_quarter, nb_semiAnnual or nb_year.

- **frequency** ↑

Frequency of data:
1 : yearly (also used for integers)
2 : semiannually
4 : quarterly
12 : monthly
52 : weekly
365 : daily

- **ignorenan** ↑

A boolean indicating if we should (1) (default) ignore NaNs when doing calculations on the dataset or (0) take NaNs into account when doing these calculations. Note that taking NaNs into account require the dataset to be expanded prior to the operation and will consequently take more time. Also, this will behave like a nb_ts calculation.

- **indicator** ↑

A boolean indicating if either (1) the locations where we do have an observation should be saved as 1 or (0) the locations where we do not have an observation should be saved as 1. This affects the memory required to save the sparse <locations> double. By default this will be set to the "best" option, that is the option that requires the least memory.

indicator = 1 : observation = 1, missing value = 0
Indicator = 0 : observation = 0, missing value = 1

- **localVariables** ↑

A nb_struct with the local variables of the nb_ts object.
I.e. a field with name 'test' can be reach with the string input %#test. Only for dates and vintage inputs.

Inherited from superclass NB_DATASOURCE

- **locations** ↑

Sparse logical keeping track of where we have observations or not. NaN is counted as a missing observation.

- **numberOfDatasets** ↑

Number of datasets stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **numberOfObservations** ↑

Number of observation of the data stored in the object. As a double.

- **numberOfVariables** ↑

Number of variables stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **startDate** ↑

The start date of the data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_week, nb_month, nb_quarter, nb_semiAnnual or nb_year.

- **userData** ↑

Add user data. Can be of any type.

Inherited from superclass NB_DATASOURCE

- **variables** ↑

Variables of the object. As a cellstr.

Inherited from superclass NB_DATASOURCE

Methods:

- abs
- acosh
- acoth
- acsch
- addPageCopies
- addPrefix
- appendFromAnotherPage
- asecd
- asind
- atan
- breakLink
- callstat
- corr
- cosh
- coth
- csc
- cumnansum
- dates
- demean
- egrowth
- exp
- extMethod
- getClassOfData
- getMethodCalls
- getSource
- growth
- head
- acos
- acot
- acsc
- addDataset
- addPages
- addVariable
- asCell
- asech
- asinh
- atand
- callfun
- ceil
- cos
- cot
- cov
- cscd
- cumprod
- deleteMethodCalls
- disp
- empty
- expand
- floor
- getCreatedVariables
- getRealEndDate
- getSourceList
- gt
- histogram
- acosd
- acotd
- acscd
- addDatasets
- addPostfix
- and
- asec
- asin
- assignNaN
- atanh
- callop
- conj
- cosd
- cotd
- createVariable
- csch
- cumsum
- deleteVariables
- double
- eq
- expm1
- ge
- getLocInd
- getRealStartDate
- getVariable
- hasVariable
- horzcat

- initialize
- isCrossSectional
- isTimeseries
- isfinite
- keepPages
- log
- log2
- max
- merge2Series
- not
- or
- plot
- real
- rename
- round
- seed
- setMethodCalls
- sin
- skewness
- sqrt
- struct
- tail
- tanh
- uminus
- uplus
- zeros
- interpolate
- isDistribution
- isUpdateable
- isnan
- kurtosis
- log10
- lt
- mean
- nan
- objSize
- packing
- pow2
- reallog
- renameMore
- saveDataBase
- sech
- setValue
- sind
- sort
- std
- structure2Dataset
- tan
- toStructure
- unstruct
- var
- isBoolean
- isDouble
- isempty
- isscalar
- le
- log1p
- map2Distribution
- merge
- ne
- ones
- permute
- rand
- realsqrt
- reorder
- sec
- setInfinite2NaN
- setZero2NaN
- sinh
- sortProperty
- stopSorting
- sum
- tand
- tonb_ts
- update
- window

► **abs** ↑

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = abs(in);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acos** ↑

obj = acos(obj)

Description:

Take acos of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acos(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acosd** ↑

obj = acosd(obj)

Description:

acosd(obj) is the inverse cosine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acosh** ↑

obj = acosh(obj)

Description:

acosh(obj) is the inverse hyperbolic cosine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acot** ↑

obj = acot(obj)

Description:

Take acotangent of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acot(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acotd** ↑

obj = acotd(obj)

Description:

acotd(obj) is the inverse cotangent, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acotd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acoth** ↑

obj = acoth(obj)

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acoth(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsc** ↑

obj = acsc(obj)

Description:

Take inverse csc of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acsc(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acscl** ↑

obj = acscl(obj)

Description:

acscl(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acscd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsch** ↑

obj = acsch(obj)

Description:

acsch(obj) is the inverse hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acsch(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **addDataset** ↑

obj = addDataset(obj,dataset,nameOfDataset,startDate,locations,...
indicator,variables)

Description:

Makes it possible to add a dataset to an existing nb_bd object

Caution : If more than one dataset is added to an object the link to the data source is broken!

Input:

- obj : An object of class nb_bd
- dataset : Either a string with the name of excel spreadsheet or a .mat file (no extension needed), or a numerical matrix.
- nameOfDataset : Must be a string with the dataset name. If not given (or given as '') the default name Database<jj> is given. Where jj is the number of added datasets of the object, including dataset you add with this method.
- dates : A cell array of string with the dates you want to add. Only needed when numerical matrix of data is added to the object. Must have the same size as the data you add.
- variables : A cell array of string with the names of the variables you want to add. Only needed when numerical matrix of data is added to the object. Must have the same size as the data you add.

Output:

- obj : The object itself with the added dataset.

Written by Per Bjarne Bye

► addDatasets ↑

```
obj = addDatasets(datasets,nameOfDatasets,dates,variables)
```

Makes it possible to add more datasets to a existing nb_bd object

Input:

Same input as of the constructor but only cell arrays are supported for the inputs 'datasets' and 'nameOfDatasets'. 'nameOfDatasets' could be given as a empty cell; {};

Output:

- obj : An object of class nb_bd, now including the new datasets provided

Examples:

```
obj = addDatasets({'2019Q1','2019Q2'},{},{'First'},...  
{'Var1','Var2'})
```

Written by Kenneth S. Paulsen

► **addPageCopies** ↑

Description:

Add copies of the data of an object of class nb_bd and append it as new datasets of the object. Only possible if an object has only one page.

Input:

- obj : An object of class nb_bd
- num : Number of copies to be appended

Output:

- obj : An object of class nb_bd with the added copies of the first dataset in the object.

Examples:

```
obj = nb_bd([2,2],'test','2012Q1',{'Var1','Var2'});  
obj = addPageCopies(obj,2);
```

Written by Kenneth S. Paulsen

► **addPages** ↑

```
obj = addPages(obj,DB)
```

Description:

Add all pages from a different nb_bd object with the current one

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_bd
- DB : An object of class nb_bd
- varargin : Optional number of objects of class nb_bd

Output:

- obj : An object of class nb_bd with all the datasets from the objects obj and varargin.

Be aware: If the added datasets does not contain the same variables or dates, the data of the nb_bd object with the tightest window will be expanded automatically to include all the same dates and variables. (The added data will be set as nan when expanded to full representation.)

Examples:

```
obj = addPages(obj,DB);
```

Written by Per Bjarne Bye

► **addPostfix** ↑

```
obj = addPostfix(obj,postfix)
obj = addPostfix(obj,postfix,vars)
```

Description:

Add a postfix to all the variables in the nb_ts, nb_cs or nb_data object, or a a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- postfix : The postfix to add to all the variables of the object. Must be a string
- vars : A cellstr of the variable to add the prefix to. Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with the postfix added to the variables selected

Examples:

```
obj = obj.addPostfix('_NEW');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **addPrefix** ↑

```
obj = addPrefix(obj,prefix)
obj = addPrefix(obj,prefix,vars)
```

Description:

Add a prefix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- prefix : The prefix to add to all the variables of the object.
Must be a string
- vars : A cellstr of the variable to add the prefix to.
Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with a prefix added to the selected variables

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **addVariable ↑**

```
obj = addVariable(obj,startDateOfData,Data,Variable)
```

Description:

Add a new variable to a existing nb_bd object (including all pages)

Input:

- obj : An object of class nb_bd
- startDate : The start date of the data of the added variable. Must be a string or a date object. Cannot be outside the window of the dates of the object.
- data : The data of the added variable. Cannot be outside the window of the data of the object. Must be a n x 1 or 1 x n double.
- varname : A string with the added variable name. Cannot be the same as a existing variable. (Then use setValue instead)

Output:

- obj : An object of class nb_bd with the added variable

Examples:

```
obj = addVariable(obj,'2012Q1',[2;2;2;2],'newVariable');
```

Written by Per Bjarne Bye

► and ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the and operator has evaluated all the data elements of the object.
The data will be a logical matrix

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► appendFromAnotherPage ↑

```
obj = appendFromAnotherPage(obj,page)
```

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_dataSource object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_dataSource object.

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **asCell ↑**

```
cellMatrix = asCell(obj,page,strip)
```

Description:

Return the nb_bd objects data as a cell matrix.

Input:

- obj : An object of class nb_bd
- page : Which pages of the expanded dataset should be transformed to a cell. Must be a double (vector) with the indicies.
- strip : Set to false to not strip all the nan observations.

Output:

- cellMatrix : The objects data transformed to a cell. (With types and variable names)

Examples:

```
obj      = nb_bd([2,2],'test','2000Q1',{'Var1','Var2'});  
cellMatrix = obj.asCell();
```

Written by Per Bjarne Bye

► **asec ↑**

```
obj = asec(obj)
```

Description:

Take inverse sec of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asecd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asech** ↑

```
obj = asech(obj)
```

Description:

```
asech(obj) is the inverse hyperbolic secant of the elements of obj
```

Input:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Output:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Examples:

```
out = asech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asin ↑**

```
obj = asin(obj)
```

Description:

Take inverse sin of the data stored in the nb_ts, nb_cs or nb_data object

Input:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Output:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asind ↑**

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **assignNan** ↑

```
obj = expand(obj,variables,value)
```

Description:

Assign all the nan observation to the value input.

Input:

- obj : An object of class nb_dataSource.
- variables : A char or a cellstr with the variables to assign nan values.
- value : A scalar number.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj      = nb_ts.rand('2012Q1',10,2,3);
obj(1:2,1,:) = nan;
obj      = assignNan(obj,'Var1',0)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the inverse tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees, of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **breakLink** ↑

```
obj = breakLink(obj)
```

Description:

Break link to source

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callfun ↑**

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb_dataSource object(s). The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb_dataSource.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nb_dataSource.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb_dataSource objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

```
- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource object. The rest will be given as return by the func function when applied to the data of the object.
```

Examples:

```
d      = nb_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)
```

See also:

[nb_bd.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callop** ↑

```
obj = callop(obj,another,func)
obj = callop(obj,another,func,type)
```

Description:

Call mathematical operator on the data of the object. In contrast to calling the operator itself on the object, this method does not try to match the same variables from the two inputs, but instead operate in the same way as mathematical operators do on double matrices (bsxfun). The time domain is matched though!

Caution: The following operators are supported:

```
> @plus or 'plus'
> @minus or 'minus'
> @times or 'times'
> @rdivide or 'rdivide'
> @power or 'power'
```

Input:

```
- obj      : An object of class nb.DataSource.
- another   : An object of class nb.DataSource.
- func      : One of the above listed functions.
- type      : Either:
              > 'keep'    : Keep the variable names of the object with most variables. If they have the same number
```

```

of variables, they are taken from the first
object.
> 'rename' : The new variable names represent the operation
that has been done to the separate series. So
if you divide an object with one series named
'Var1' with another object with one series
named 'Var2', the new name will be
'Var1./Var2'. (Default)

```

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource method. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```

d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,1),'''2011Q1',{'Var2'});
d3 = callop(d1,d2,@minus);

d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,2),'''2011Q1',{'Var2','Var3'});
d3 = callop(d1,d2,@minus);

```

See also:

[nb_bd.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callstat** ↑

```

obj = callstat(obj,func)
obj = callstat(obj,func,varargin)

```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s) that goes from the data of the object have more than one variable to only having one. E.g. when taking the mean over a set of variables.

Input:

- obj : An object of class nb.DataSource.
- func : A function handle (or one line char) that maps a nObs x nVar x nPage double to nObs x 1 x nPage double, or that maps a nObs x nVar x nPage double to nObs x nVar x 1 double.

Optional input:

- 'name' : Set the name of the new variable, as a one line char.
Default is to use the str2func on the provided function handle.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- obj : An object of class nb_dataSource with only one variable.

Examples:

```
d = nb_ts.rand('2012Q1',10,3);
d1 = callstat(d,@(x)mean(x,2),'name','mean')
```

See also:

[nb_bd.mean](#), [nb_bd.std](#), [nb_bd.var](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ceil ↑**

```
obj = ceil(obj)
```

Description:

`ceil(obj)` rounds the elements of `obj` to the nearest integers towards infinity.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = ceil(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **conj** ↑

```
obj = conj(obj)
```

Description:

`conj(obj)` is the complex conjugate of the elements of `obj`.
For a complex element `x` of `obj`, $\text{conj}(x) = \text{REAL}(x) - i\text{IMAG}(x)$.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = conj(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **corr** ↑

```
obj = corr(obj,varargin)
```

Description:

Calculate the correlation of the timeseries of the `nb_bd` object

Caution: Calculates the correlation matrix of the variables of the `nb_bd` object, if no optional inputs are given. If 'lags' is given this function calculates the correlation with the number of wanted lags for each variable with itself. (And only with itself)

If the `<ignorenan>` property of the `nb_bd` object is set to true, missing values are not accounted for. This is equal to setting the 'robust' argument to true in the `<corr>` method of `nb_ts`.

Input:

- `obj` : An object of class `nb_bd`

Optional input ('propertyName',PropertyValue):

- 'lags' : An integer with the number of lags you want

to calculate the correlation for. (Here for each variable)

- 'type' : 'pairwise' or 'default'. 'pairwise' calculate the correlation matrix for the selected lag or lead, while 'default' calculates the correlation with itself for lags from 1 to lags and the same for lead.

Output:

- obj : An nb_cs object with the wanted correlation

Examples:

```
corrMatrix = corr(obj);  
corrMatrix = corr(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► **cos** ↑

```
obj = cos(obj)
```

Description:

Take cos of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

`cosd(obj)` is the cosine of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = cosd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cosh** ↑

```
obj = csch(obj)
```

Description:

`cosh(obj)` is the hyperbolic cosine of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cot** ↑

```
obj = cot(obj)
```

Description:

Take cotangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = cot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

cotd(obj) is the cotangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **coth** ↑

```
obj = coth(obj)
```

Description:

`coth(obj)` is the hyperbolic cotangent of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = coth(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cov ↑**

```
obj = cov(obj,varargin)
```

Description:

Calculate the covariances of the timeseries of the `nb_bd` object

Caution: Calculates the covariance matrix of the variables of the `nb_bd` object, if no optional inputs are given. If 'lags' is given this function calculates the covariance with the number of wanted lags for each variable with itself. (And only with itself)

If the `<ignorenan>` property of the `nb_bd` object is set to true, the dataset any row with a NaN will be removed and calculations will be done on that data for the whole vcv-matrix. If `<ignorenan>` is false, NaNs will be trimmed to make data vectors conform when calculating covariances, but for variances, max number of rows is kept.

Input:

- `obj` : An object of class `nb_bd`

Optional input ('propertyName',PropertyValue):

- 'lags' : An integer with the number of lags you want to calculate the covariance for. (Here for each variable)
- 'type' : 'pairwise' or 'default'. 'pairwise' calculate the covariance matrix fro the selected lag or lead, while 'default' calculates the covariance with itself for lags from 1 to lags and the same for lead.

Output:

- obj : An nb_cs object with the wanted covariances

Examples:

```
corrMatrix = cov(obj);  
corrMatrix = cov(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► createVariable ↑

```
obj = createVariable(obj,nameOfNewVariable,expression)  
obj = createVariable(obj,nameOfNewVariable,expression,parameters,varargin)
```

Description:

Create variable(s) and add them to the nb_bd object dataset(s)

- This function only support methods of the class nb_math_ts for the expressions. Type the following in the command window;

```
methods(nb_math_ts)
```

Input:

- obj : An object of class nb_bd
- nameOfNewVariable : The created variable names as a cell array or a string.
Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the data of all the created variables.
As a cell array or a string.
Must have the same size as the 'nameOfNewVariables' input.
- parameters : A struct with the parameters that can be used in the expressions. Caution: They will be added as series, so you need to use elementwise operators (.*, ./, .^)!

Optional input:

- 'overwrite' : true or false. Set to true to allow for overwriting of variables already in the data of the object. Default is false.

- 'warning' : true or false. Set to true to give warning if expressions fail (instead of error). Will result in series having all nan value, when warning is thrown.

Output:

- obj : An nb_bd object with the created variable(s) added.

Examples:

```
obj = obj.createVariable('var3','var1./var2');  
obj = obj.createVariable({'var3','var4',...},...  
{'var1./var2','var2./var1',...});
```

See also:

[nb_math_ts](#)

Written by Per Bjarne Bye

► **csc** ↑

obj = csc(obj)

Description:

Take csc of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cscd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csch** ↑

```
obj = csch(obj)
```

Description:

csch(obj) is the hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cumnansum** ↑

```
obj = cumnansum(obj)
```

Description:

Cumulativ sum of the series of the object. Ignoring nan values.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumnansum(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumprod** ↑

```
obj = cumprod(obj,dim)
```

Description:

Cumulativ product of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : A nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative product of the old objects 'data' property.

Examples:

```
obj = cumprod(obj);
obj = cumprod(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumsum(obj);  
obj = cumsum(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **dates** ↑

```
allDates = dates(obj)  
allDates = dates(obj,format,type)
```

Description:

Get all the dates of the object. As a cellstr. Dates where all variables are nan are stripped.

Input:

- obj : An object of class nb_bd
- format : See the method toDates of the nb_year, nb_semiAnnual, nb_quarter, nb_month, nb_week or nb_day classes.
- type : Either 'stripped' or 'full' (default). If 'stripped' is given all dates where all observations of the data is nan is removed, otherwise all dates are returned.

Output:

- allDates : A cellstr array of the dates of the object

Examples:

```
allDates = obj.dates();  
allDates = obj.dates('xls');
```

Written by Kenneth S. Paulsen

► deleteMethodCalls ↑

```
obj = deleteMethodCalls(obj,c)
```

Description:

Delete some method calls of the object. The object needs to be updatable.

Caution: To set method calls use setMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- sourceNr : The source index to delete the methods from
- methodNrs : A index with the methods to delete.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

See also:

[setMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► deleteVariables ↑

```
obj = deleteVariables(obj,deletedVar)
```

Description:

```
Delete variables from the current nb_bd
```

If some of the variables you specify in deletedVar does not exist in the obj, this will not have anything to say for the obj. (The variables cannot be deleted, because the variables do not exist.)

This method overrides the implementation in nb_dataSource.

Input:

- obj : An object of class nb_bd
- deletedVar : Must be a char or a cellstr

Delete all the variable names, given in the char array or cellstr array deletedVar, of current the object.

NB! If deletedVar is just one string and include a * as the last letter, this function will delete all the variable names, from the object, that start with the letters which is in front of the * in the string.

i.e. obj = deleteVariables(obj,'DPQ*');

Deletes all the variables in the obj which has variable names starting with 'DPQ'.

If deletedVar is just one string and include a * as the first letter, this function will delete all the variable names, from the object, that include the letters which follows in the string.

i.e. obj = deleteVariables(obj, '*shift');

Deletes all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_bd object with the variables specified in deletedVar deleted.

Examples:

```
obj = nb_bd([2,2],'test','2012',{'Var1','Var2'});  
obj = deleteVariables(obj, {'Var1','Var2'});  
obj = deleteVariables(obj, char('Var1','Var2'));  
obj = deleteVariables(obj, obj.variables);
```

Written by Per Bjarne Bye

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_dataSource object.
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_dataSource object.

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATASOURCE

► **disp ↑**

disp(obj)

Description:

Display object (In the command window)

Input:

- obj : An object of class nb_bd

Output:

The dataset displayed in the command window.

Examples:

obj (Note without semicolon)

Written by Per Bjarne Bye

► **double ↑**

dataOfObject = double(obj,type)

Description:

```
Get the data of the nb_bd object as a double matrix
```

This method overrides the nb_dataSource method inherited by the other data objects, as we

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- type : Either 'full' (default), 'stripped' or 'sparse'. Can also be logical true ('full') or false ('sparse').

Output:

- dataOfObject : The data of the nb_bd object.

Examples:

```
dataOfObject = double(obj)
```

Written by Per Bjarne Bye

► **egrowth ↑**

```
obj = egrowth(obj)
obj = egrowth(obj,lag)
```

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$ of all the timeseries of the nb_bd object.

Input:

- obj : An object of class nb_bd
- lag : The number of lags in the growth formula, default is 1.

Caution : The <ignorenan> property of nb_bd obj controls of NaNs will be stripped before calculation of growth or not.

Output:

- obj : An nb_bd object with the calculated timeseries stored.

Examples:

```
obj = egrowth(obj);
obj = egrowth(obj,4);
```

See also:

[growth](#), [aegrowth](#), [agrowth](#)

Written by Per Bjarne Bye

► **empty** ↑

`obj = empty(obj)`

Description:

Empty the nb_bd object

Input:

- obj : An object of class nb_bd

Output:

- obj : An empty nb_bd object

Examples:

`obj = empty(obj);`

Written by Per Bjarne Bye

► **eq** ↑

`a = eq(a,b)`

Description:

Test if the one object is equal (==) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.

- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **exp ↑**

```
obj = exp(obj)
```

Description:

Take exp of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **expand ↑**

```
obj = expand(obj,newStartDate,newEndDate,warningOff)
```

Description:

Expand the current nb_bd object with more dates and data.

Input:

```
- obj : An object of class nb_bd  
- newStartDate : The wanted new start date of the data.  
- newEndDate : The wanted new end date of the data.  
- warningOff : Give 'off' to suppress warning if no expansion  
has taken place. Both 'newStartDate' and
```

Output:

```
- obj : An nb_bd object with expanded timespan
```

Examples:

```
obj = nb_bd.rand('1901Q1',10,2);  
obj = expand(obj,'1900Q1','1903Q4');
```

Written by Per Bjarne Bye

► **expml** ↑

```
obj = expml(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, e to the `obj`.
`expml(obj)` computes `exp(obj)-1`, compensating for the roundoff in
`exp(obj)`.
(For small real `X`, `expml(X)` should be approximately `X`, whereas the
computed value of `EXP(X)-1` can be zero or have high relative error.)

Input:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Output:

```
- obj : An object of class nb_ts, nb_cs or nb_data where the  
data are equal to  $e.^obj.data - 1$ 
```

Examples:

```
obj = expml(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **extMethod** ↑

```
obj = extMethod(obj,method,variables,postfix,varargin)
```

Description:

This method can call a nb_dataSource method on a selected variables, and merge the result with the old object by adding a postfix to the new variables.

This method is the same as calling:

```
obj1 = window(obj,'','','variables');
method = str2func(method);
obj = method(obj,varargin{:});
obj1 = addPostfix(obj1,method);
obj = merge(obj,obj1);
```

But without replicating the links property unnecessary many times!

Caution : If the object is not updateable this will result in the same as the code above!

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- method : A str with the method to call. Must be a method of the nb_ts, nb_data or nb_cs class that does not change other dimensions than the second dimension!
- variables : A cellstr with the variables of the object to call the method on. Must be included in the object.
- postfix : A string with the postfix. If empty, the old variables will be replaced by the new ones.

Optional inputs:

These will be the inputs casted to the method except the object itself.

Extra for nb_ts and nb_data:

- '<start>' : The start date/obs of the function evaluation. Given as the second input to the window method call.
- '<end>' : The end date/obs of the function evaluation. Given as the third input to the window method call.

Output:

- obj : A nb_ts, nb_data or nb_cs object

Examples:

```
obj = nb_ts.rand(1,10,3);
obj = extMethod(obj,'lag',{'Var1'},'lag1',1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **floor** ↑

```
obj = floor(obj)
```

Description:

`floor(obj)` rounds the data elements of `obj` to the nearest integers towards minus infinity

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = floor(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **ge** ↑

```
a = ge(a,b)
```

Description:

Test if the one object is greater than or equal to (\geq) the other object elemetswise and return a `nb_dataSource` object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- `a` : An object of class `nb_dataSource` or a scalar number.
- `b` : An object of class `nb_dataSource`, but must be of same subclass as `a`, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **getClassOfData** ↑

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_dataSource.

Output:

- cl : Name of the class that the data of the object is stored as.

See also:

[nb_bd.isDistribution](#), [nb_bd.isDouble](#)
[nb_bd.isBoolean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getCreatedVariables** ↑

```
created = getCreatedVariables(obj)
```

Description:

Get a N x 2 table of the variables created using the createVariable method. The first column list the created variables, while the second column list the expressions.

Input:

- obj : An object that is a subclass of nb_dataSource.

Output:

- created : A N x 2 cell array. See description of this method for more.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getLocInd** ↑

[loc,ind,dataOut,numberOfDatasets] = getLocInd(dataIn)

Description:

Given a n x n double of data, return the locations of data as a sparse logical, the indicator with information if missing values is stored as a 1/0, and the data as a vector with no missing values.

Input:

- data : A n x n double with the data.

Output:

- loc : A n x n sparse logical with information of the positions.

- ind : A 1 x 1 logical. 1 if observations are stored as a 1 and missing values as a 0 in loc, and 0 otherwise.

- dataOut : A n x 1 double with only the data (no missing values).

- numberOfDatasets : A 1 x 1 double. This is equal to the third dimension of the data as a sparse logical can only be 2D. The third dimension is preserved by stacking locations horizontally.

Examples:

```
data = [1,2,3;1,NaN,4;5,NaN,7];
[loc,ind,dataOut,numberOfDatasets] = getLocInd(data)
```

Written by Per Bjarne Bye

► **getMethodCalls** ↑

[s,c] = getMethodCalls(obj)

Description:

Get all method calls as a cell matrix. The object needs to be updatable to get a list of methods called on the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.
- c : A cell matrix with a table of the method called on the object and its input. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getRealEndDate ↑

```
realEndDate = getRealEndDate(obj)
realEndDate = getRealEndDate(obj,format,type)
realEndDate = getRealEndDate(obj,format,type,variables)
```

Description:

Get the real end date of the nb_ts object. I.e the last observation which is not nan or infinite.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
> 'xls'
> 'pprnorsk' or 'mprnorwegian'
> 'pprengelsk' or 'mprenglish'
> 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
> 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.

Output:

- realEndData : The last observation of the object which is not nan or infinite. As a string

Examples:

```
realEndDate = obj.getRealEndDate();  
realEndDate = obj.getRealEndDate('pprnorsk');
```

Written by Kenneth S. Paulsen

► **getRealStartDate ↑**

```
realStartDate = getRealStartDate(obj)  
realStartDate = getRealStartDate(obj,format,type,variables)
```

Description:

Get the real start date of the nb_bd object. I.e the first observation which is not nan or infinite.

Input:

- obj : An object of class nb_bd
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.

Output:

- realStartDate : The first observation of the object which is not nan or infinite. As a string

Examples:

```
realStartDate = obj.getRealStartDate();  
realStartDate = obj.getRealStartDate('pprnorsk');
```

Written by Kenneth S. Paulsen

► **getSource** ↑

```
sourceType = getSource(obj)
sourceType = getSource(obj,type)
```

Description:

Get source type.

Input:

- obj : An object of class nb_dataSource.
- type : Give 'program' to get the general source. Default is to give the specific type of source.

Output:

- sourceType : A one line char with the source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSourceList** ↑

```
s = getSourceList(obj)
```

Description:

Get the source list of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getVariable** ↑

```

variableData = getVariable(obj,variableName)
variableData = getVariable(obj,variableName,startDate, ...
                           endDate, pages)
variableData = getVariable(obj,type,variableName)
variableData = getVariable(obj,type,variableName,startDate, ...
                           endDate, pages)

```

Description:

Get the data of a variable, as a double

Input:

- obj : An object of class nb_ts
- type : Either 'stripped' or 'full' (default). If 'stripped' is given all dates where all observations of the data is nan is removed, otherwise all dates are returned.
- variableName : The variable you want the data of. Must be given as a string (name of variable) or a cellstr.
- startDate : Start date of the return data of the given variable. As a string or an object which is of a subclass of nb_date
- endDate : End date of the return data of the given variable. As a string or an object which is of a subclass of nb_date
- pages : Pages of the return data of the given variable. As a double or logical array, e.e. [1:2] or [1,3,5].

Output:

- variableData : The data of the variable you have given. Return [] if not found.

Examples:

```

variableData = getVariable(obj,'Var1');
variableData = getVariable(obj,'Var1','2012Q1','2012Q4');
variableData = getVariable(obj,'full','Var1');
variableData = getVariable(obj,'full','Var1','2012Q1');
variableData = getVariable(obj,'full','Var1','2012Q1','2012Q4');
variableData = getVariable(obj,'full','Var1','2012Q1','2012Q4',[1:3]);

```

Written by Kenneth S. Paulsen

► **growth ↑**

```

obj = growth(obj)
obj = growth(obj,lag,stripNaN)

```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_bd
- lag : The number of lags in the approx. growth formula, default is 1.

Output:

- obj : An nb_bd object with the calculated timeseries stored.

Examples:

```
obj = growth(obj);
obj = growth(obj, 4);
```

See also:

[egrowth](#), [aegrowth](#), [agrowth](#)

Written by Per Bjarne Bye

► **gt** ↑

```
a = gt(a,b)
```

Description:

Test if the one object is greater then ($>$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a > b;  
obj = 2 > b;  
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hasVariable** ↑

```
[ret,location] = hasVariable(obj,variable)
```

Description:

Test if an nb_cs object has a given variable

Input:

- obj : An object of class nb_cs
- variable : The variable name as a string. Can also be a cellstr with more variables.

Output:

- ret : 1 if found otherwise 0
- location : The location of the variable if found. Otherwise [].

Examples:

```
ret = obj.hasVariable('Var1')  
[ret,location] = hasVariable(obj,'Var1')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **head** ↑

```
head(obj,nRows,page)
```

Description:

Display the first n rows of a nb_bd object.

Input:

- obj : An nb_bd object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_bd](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **histogram** ↑

```
data          = histogram(obj)
[data,plotter] = histogram(obj,M)
```

Description:

Create a histogram of a object containing only one variable and one page!

Input:

- obj : A nb_dataSource object with size N x 1 x 1.
- M : The number of bins of the histogram.

Output:

- data : A nb_cs object with size M x 1 x 1.
- plotter : A nb_graph_cs object with the histogram plotted.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **horzcat** ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation of nb_bd objects. ([a,b]) This is only possible if the different objects contain different variables or have variables with the same values (start and/or end dates can differ). Uses the merge method.

Input:

- a : An object of class nb_bd
- varargin : Optional number of nb_bd objects

Output:

- obj : An nb_bd object with all the variables from the different nb_bd object merged into one dataset

Examples:

```
obj = [a,b];  
obj = [a,b,c];
```

See also:

[merge](#)

Written by Kenneth S. Paulsen

► **initialize ↑**

```
obj = nb_bd.initialize(data,nameOfDatasets,dates,variables,sorted)
```

Description:

Initialize a nb_db object with a stripped set of dates and its matching data points.

Input:

- data : A r x c x p double, logical or nb_distribution array.
- nameOfDatasets : Either a one line char or a 1 x p cellstr.
- dates : A 1 x r or r x 1 nb_date array.
- variables : A 1 x c cellstr.
- sorted : true or false. Default is true.

Output:

- obj : A nb_bd object.

See also:

[nb_bd](#)

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_data object. Will discard leading and trailing nan values.

Input:

- data : An nb_ts, nb_cs or nb_data object.
- method :
 - 'nearest' : Nearest neighbor interpolation
 - 'linear' : Linear interpolation. Default
 - 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - 'pchip' : Shape-preserving piecewise cubic interpolation
 - 'cubic' : Same as 'pchip'
 - 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : An nb_ts, nb_cs or nb_data object with the interpolated data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isBoolean** ↑

```
ret = isBoolean(obj)
```

Description:

Check if the data of the nb_dataSource object is of class logical (boolean, i.e. true or false).

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isCrossSectional ↑**

ret = isCrossSectional(obj)

Description:

Test if this object is a cross sectional data object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_cs, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isDistribution ↑**

ret = isDistribution(obj)

Description:

Check if the data of the nb_dataSource object is of class nb_distribution

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isDouble ↑**

```
ret = isDouble(obj)
```

Description:

Check if the data of the nb_dataSource object is of class double (numeric)

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isTimeseries** ↑

```
ret = isTimeseries(obj)
```

Description:

Test if a nb_dataSource object is a time series object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_ts, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isUpdateable** ↑

```
ret = isUpdateable(obj)
```

Description:

Test if this object is updateable.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : 1 if updateable, otherwise 0.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isempty** ↑

ret = isempty(obj)

Description:

Test if a nb_ts, nb_cs or nb_data object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

ret = isempty(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isfinite** ↑

obj = isfinite(obj)

Description:

Test if each element of a nb_ts, nb_cs or nb_data object is finite.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_ts, nb_cs or nb_data object is nan.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isscalar** ↑

```
ret = isscalar(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is a scalar. Will always return 0, as an object of class nb_ts, nb_cs or nb_data is not a scalar.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : false

Examples:

```
0 = isscalar(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepPages** ↑

```
obj = keepPages(obj, pages)
```

Description:

Keep pages of the current nb_ts object

Input:

- obj : An object of class nb_ts
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty a empty nb_ts object is returned.

Can also be a cellstr with the names to keep, or a logical array with length equal to the number of pages (datasets).

Output:

- obj : An nb_ts object with the pages specified in the input are kept.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **kurtosis** ↑

```
obj = kurtosis(obj, flag, outputType, dimension)
```

Description:

Calculate the kurtosis of each timeseries. The result will be an object of class nb_bd where all the non-nan values of all the timeseries are being set to their kurtosis.

The output can also be set to be a double or a nb_cs object consisting of the kurtosis values of the data.

Input:

- obj : An object of class nb_bd
- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N
 - Where N is the sample length.
- outputType :
 - > 'nb_bd' : The result will be an object of class nb_bd where all the non-nan values of all the timeseries are being set to their kurtosis.
 - > 'double' : Get the kurtosis values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the kurtosis values as nb_cs object. The kurtosis will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the kurtosis over.

Caution: the `<ignorenan>` property of obj controls the behavior of how to handle NaNs. This is equal to the 'notHandleNaN' optional input of the std method of the nb_ts class.

Output:

- obj : See the input outputType for available options.

Examples:

```
nb_bdObj = kurtosis(obj);
double    = kurtosis(obj,0,'double');
nb_csObj = kurtosis(obj,0,'nb_cs');
```

Written by Per Bjarne Bye

► le ↑

```
a = le(a,b)
```

Description:

Test if the one object is less than or equal to (\leq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► log ↑

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on logs.

Examples:

```
obj = log(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_DATASOURCE
```

► **log10** ↑

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on (base 10) logs.

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **log2** ↑

```
obj = log2(obj)
```

Description:

log2(obj) is the base 2 logarithm of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **lt** ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less than (<) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a < b;  
obj = 2 < b;  
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► map2Distribution ↑

```
obj = map2Distribution(obj,dist)  
obj = map2Distribution(obj,dist,varargin)
```

Description:

Map observation to the distributions given by dist.

The current distribution of the data is estimated using a kernel density estimator. The data must be strongly stationary. I.e. the unknown distribution from where the data has been drawn must not depend on time.

Input:

- obj : An object of class nb_dataSource.
- dist : An object of class nb_distribution with size 1 x size(obj,2).

Optional inputs:

- See the optional inputs of the nb_distribution.estimate function.

Output:

- obj : An object of class nb_dataSource.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **max** ↑

```
obj = max(obj,outputType,dimension)
```

Description:

Calculate the max of each timeseries. The result will be an object of class nb_bd where all the non-nan values of all the timeseries are being set to their max.

The output can also be set to be a double or a nb_cs object consisting of the max values of the data.

Input:

- obj : An object of class nb_bd
- outputType :
 - > 'nb_bd' : The result will be an object of class nb_bd where all the non-nan values of all the timeseries are being set to their max.
 - > 'double' : Get the max values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the max values as nb_cs object. The max will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the max over.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_bdObj = max(obj);
double   = max(obj,'double');
nb_csObj = max(obj,'nb_cs');
```

Written by Per Bjarne Bye

► **mean** ↑

```
obj = mean(obj,outputType,dimension,varargin)
```

Description:

Calculate the mean of each timeseries. The result will be an object of class nb_bd where all the non-nan values of all the timeseries are being set to their mean.

The output can also be set to be a double or a nb_cs object consisting of the mean values of the data.

Input:

- obj : An object of class nb_bd
- outputType :
 - > 'nb_bd' : The result will be an object of class nb_bd where all the non-nan values of all the timeseries are set to the mean.
 - > 'double' : Get the mean values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mean values as nb_cs object. The mean will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calcualate the mean over.

Caution: the <ignorenan> property of obj controls the behavior of how to handle NaNs. This is equal to the 'notHandleNaN' optional input of the mean method of the nb_ts class.

Output:

- obj : See the input outputType for available options.

Examples:

```
nb_bdObj = mean(obj);  
double    = mean(obj,'double');  
nb_csObj = mean(obj,'nb_cs');
```

Written by Per Bjarne Bye

► **merge ↑**

```
obj = merge(obj,DB)
```

Description:

Merge another nb_bd object with the current one

Input:

- obj : An object of class nb_bd
- DB : An object of class nb_bd

Output:

- obj : An object of class nb_bd where the datasets from the DB object are merged with the data of the obj object.

Caution:

- > It is possible to merge datasets with the same variables as long as they are representing the same data or have different types.
- > If the datasets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they can merge.
- > It is not possible to merge two bd objects of different frequencies. Use nb_ts instead.

Examples:

```
obj = nb_bd([2,2; 2,2],'test','2000Q1',{'Var1','Var2'});  
DB = nb_bd([2,1; 2,2],'test','2000Q2',{'Var1','Var3'});  
obj = obj.merge(DB)
```

Written by Per Bjarne Bye

► merge2Series ↑

```
obj = merge2Series(obj,var1,var2,new,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_dataSource.
- var1 : A one line char with the name of the base series.

- var2 : A one line char with the name of the appended series.
- new : Name of the new variable. If given as empty there will not be created a new series, but instead the var1 variable will be set to the new merged series. Default is empty.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_dataSource.

Examples:

```

obj           = nb_ts.rand('2012Q1',10,2,3);
obj(end-1:end,1,:) = nan;

obj1 = merge2Series(obj,'Var1','Var2','Var3')
obj2 = merge2Series(obj,'Var1','Var2')
obj3 = merge2Series(obj,'Var1','Var2','','diff','nLags',1)
obj4 = merge2Series(obj,'Var1','Var2','','diff','nLags',4)
obj5 = merge2Series(obj,'Var1','Var2','','level','date','2014Q1')

```

See also:

[nb_ts.merge](#), [nb_data.merge](#), [nb_cs.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► [nan ↑](#)

```

obj = nb_bd.nan()
obj = nb_bd.nan(start,obs,vars,pages,sorted)

```

Description:

Create an nb_bd object with all data set to nan. Notice, this might not make much sense as the raison d'être for this class is to strip NaNs from the data. The resulting object will be an empty nb_bd object, but with locations and indicator preserved in the object.

Input:

- start : The start date of the created nb_bd object.
- obs : The number of observations of the created nb_bd object.
- vars : Either a cellstr with the variables of the nb_bd object, or a scalar with the number of variables of the nb_bd object.
- pages : Number of pages of the nb_bd object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_bd object.

Examples:

```
obj = nb_bd.nan();
obj = nb_bd.nan('2012Q1',10,['Var1','Var2','Var3']);
obj = nb_bd.nan('2012Q1',2,2,10);
```

See also:

[nb_bd](#), [nb_bd.zeros](#), [nb_bd.ones](#)

Written by Per Bjarne Bye

► [ne ↑](#)

a = ne(a,b)

Description:

Test if the one object is not equal to (~=) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **not** ↑

```
a = not(a)
```

Description:

The not operator (~).

Input:

- a : An object of class nb_dataSource.

Output:

- a : An object of class nb_dataSource. Where the not operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = ~a;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **objSize** ↑

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim :
 - 1 : Number of observations
 - 2 : Number of variables
 - 3 : Number of datasets (pages)

Output:

- varargout : See the examples below

Examples:

```
dim = objSize(obj)
```

Where dim is 1 x 2 double where the first element
is the number of observations/types and the second element is
the number of variables

```
[dim1, dim2] = objSize(obj)
```

```
[dim1, dim2, dim3] = objSize(obj)
```

```
dim1 = objSize(obj,1)
```

```
dim2 = objSize(obj,2)
```

```
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► ones ↑

```
obj = nb_bd.ones()  
obj = nb_bd.ones(start,obs,vars,pages,sorted)
```

Description:

Create an nb_bd object with all data set to ones.

Input:

- start : The start date of the created nb_bd object.
- obs : The number of observations of the created nb_bd object.
- vars : Either a cellstr with the variables of the nb_bd object, or a scalar with the number of variables of the

nb_bd object.

- pages : Number of pages of the nb_bd object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_bd object.

Examples:

```
obj = nb_bd.ones();
obj = nb_bd.ones('2012Q1',10,['Var1','Var2','Var3']);
obj = nb_bd.ones('2012Q1',2,2,10);
```

See also:

[nb_bd](#), [nb_bd.nan](#), [nb_bd.zeros](#)

Written by Per Bjarne Bye

► [or](#) ↑

a = or(a,b)

Description:

The and operator (|).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nbDataSource.
- b : An object of class nbDataSource, but must be of same subclass as a!

Output:

- a : An object of class nbDataSource. Where the or operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

a = a | b;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **packing** ↑

```
obj = packing(obj,packages,varargin)
```

Description:

Package variables by applying the wanted function. Default is summin (sum).

Input:

- obj : An object of class nb_ts.
- packages : A 2 x N cell matrix with groups of variables and the names of the groups. If you have not listed a variable it will be grouped in a group called 'Rest' (as long as 'includeRest' is not set to false).

Must be given in the following format:

```
{'group_name_1','...','group_name_N';  
 name_1 ,...,name_N}
```

Where name_x must be a string with a name of the variable or a cellstr array with the variable names. E.g 'Var1' or {'Var1','Var2'}.

Optional inputs:

- 'includeRest' : Give false to not include the 'Rest' variable, which will represent all the variable that has not been packed.
- 'func' : Either a function_handle or a one line char with the name of the function to use. The first input to this function is a nObs x nVar double, the second is the dimension (which is always 2). Examples; 'sum', 'prod', @sum or @prod.

Output:

- obj : A object of class nb_ts.

Examples:

```
data      = nb_ts.rand('2012Q1',10,4);  
packages = {'group1','group2';  
            {'Var1','Var2'},{'Var3'}};  
  
dataP1 = packing(data,packages)  
dataP2 = packing(data,packages,'includeRest',false)  
dataP3 = packing(data,packages,'func',@prod)
```

See also:

[nb_ts.createVariable](#), [nb_data.createVariable](#), [nb_cs.createVariable](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **permute** ↑

```
obj = permute(obj,variables)
```

Description:

Switch the second and third dimension of the nb_ts, nb_cs or nb_data object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- variables : A cellstr with size 1 x numberOfDatasets. Default is to use dataNames property. I.e. when not provided or if empty.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **plot** ↑

```
plot(obj,varargin)
plotter = plot(obj,varargin)
```

Description:

Plot the data of the nb_dataSource object.

Caution: If the data of the object is of class nb_distribution this method will redirect to plotDist.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- plotType : A string with either:
 - > 'graph' (Default)

```

> 'graphSubPlots'

> 'graphInfoStruct'

The name of the graphing method of the class
nb_graph_ts, nb_graph_cs or nb_graph_data.

Caution: This is an optional input. If not given it is
assumed that the optional input pairs are starting from the
second input to this function.

- varargin : Same as the input to the method set() of the
  nb_graph_ts, nb_graph_cs or nb_graph_data class.

I.e. ..., 'inputName', inputValue, ...

Caution: If the data of the object is of class
  nb_distribution, see nb_ts.plotDist for the
optional input pairs supported.

```

Output:

Plotted output

Examples:

```

plot(obj)

plot(obj,'graphSubPlots')

plot(obj,'','startGraph','2008Q1','endGraph','2011Q2')

same as

plot(obj,'graph','startGraph','2008Q1','endGraph','2011Q2')

```

See also:

[nb_graph_ts](#), [nb_ts.plotDist](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **pow2 ↑**

obj = pow2(obj)

Description:

pow2(obj) raises the number 2 to the power of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **rand** ↑

```
obj = nb_bd.rand(start,obs,vars,pages,dist,sorted)
```

Description:

Create an nb_bd object with all data set to a random number.

Input:

- start : The start date of the created nb_bd object.
- obs : The number of observation of the created nb_bd object.
- vars : Either a cellstr with the variables of the nb_bd object, or a scalar with the number of variables of the nb_bd object.
- pages : Number of pages of the nb_bd object.
- dist : A nb_distribution object to draw from.
- sorted : true or false. Default is true.

Output:

- obj : An nb_bd object.

Examples:

```
obj = nb_ts.rand('1',10,['Var1','Var2']);  
obj = nb_ts.rand('2012Q1',2,2);  
obj = nb_ts.rand('2012Q1',2,2,10,nb_distribution('type','normal'));
```

See also:

[nb_bd](#)

► **real** ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part of the elements in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **reallog** ↑

```
obj = reallog(obj)
```

Description:

Take the real natural logarithm of the data stored in the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = reallog(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **realsqrt** ↑

```
obj = realsqrt(obj)
```

Description:

`realsqrt(obj)` is the square root of the elements of `obj`.
An error is produced if `obj` contains negative elements.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = realsqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **rename** ↑

```
obj = rename(obj,type,oldName,newName)
```

Description:

Makes it possible to rename variables and datasets. For objects of class `nb_cs` types is also possible to rename.

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variable', but not when the input is 'dataset' or 'types'.

Input:

- `obj` : An object of class `nb_dataSource`

- `type` : Either:
- 'variable' : To rename a variable(s)
- 'dataset' : To rename a dataset(s) (The only input that is supported for `nb_cell`)
- 'types' : To rename a type(s) (only `nb_cs`)

- `oldName` : Case 1;

> The old variable/dataset/type name, as a string

Case 2:

> The string which should be replaced in all the variable names of the database. And reorder things afterwards. This input must end with a *.

- newName : Case 1;

> The new variable/dataset/type name, as a string

Case 2;

> The string which should replace the old string part given by the input 'oldName'.

Output:

- obj : An nb_dataSource object with the renamed variable(s), type(s) or dataset(s).

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.rename('variable','Var1','Var3');  
obj = obj.rename('variable','Var*','N_Var');  
obj = rename(obj,'variable',{'Var1','Var2'},{'VAR1','VAR2'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **renameMore** ↑

obj = renameMore(obj,type,oldNames,newNames)

Description:

Makes it possible to rename more variables, datasets and types (only nb_cs).

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variables', but not when the input is 'datasets'.

Caution : This is the same as looping over nb_dataSource.rename, but more efficient.

Caution : * syntax is not supported as is the case for nb_dataSource.rename

Input:

- obj : An object of class nb_dataSource.
- type : Either:
 - 'variables' : To rename a variables
 - 'datasets' : To rename a datasets (The only input that is supported for nb_cell)
 - 'types' : To rename a type (Only nb_cs)
- oldNames : The old variable/dataset/type names, as a cellstr.
- newName : The new variable/dataset/type names, as a cellstr.

Output:

- obj : An nb_dataSource object where the variable(s)/dataname(s)/type(s) of the object given as input is/are renamed.

Examples:

```
obj = nb_cs([22,24;27,28],'Dataset1',{'Exp','Imp'},{'Nor','Swe'})
obj = renameMore(obj,'dataset',{'Dataset1'},{'tradeBal'});
obj = renameMore(obj,'variable',{'Nor','Swe'},{'Norway','Sweden'});
obj = renameMore(obj,'type',{'Exp','Imp'},{'Export','Import'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **reorder ↑**

```
obj = reorder(obj,newOrder,type)
```

Description:

Re-order the variables/types/datasets of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- newOrder : A cellstr with the new order of the variables/types/datasets.
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **round** ↑

```
obj = round(obj,value,startDate,endDate,variables)
```

Description:

Round to closest value. I.e. if value is 0.25 it will round to the closest 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_bd object
- value : The rounding value. As a double. Default is 1.
- startDate : The start date of the rounding. As a date string or a nb_date object. Can be empty. Default is the start date of the data.
- endDate : The end date of the rounding. As a date string or a nb_date object. Can be empty. Default is the end date of the data.
- variables : The variables to round. Can be empty. Default is to round all variables.

Output:

- obj : A nb_bd object with the rounded data.

Examples:

```
obj = nb_bd([2.6923;4.1234;9.9987],'', '2012Q1','Var',[0;0;1;0],0);  
obj = round(obj,0.25)
```

```
obj =  
  
'Time'      'Var'  
'2012Q1'    [2.7500]  
'2012Q2'    [     4]  
'2012Q3'    [    NaN]  
'2012Q4'    [     10]  
'Time'      'Var'
```

Written by Per Bjarne Bye

► **saveDataBase** ↑

```
saveDataBase(obj,varargin)
```

Description:

Save data of the object to (a) file(s)

If no optional input is given, i.e. obj.saveDataBase(). The data of the object is save down to xlsx file(s). Each dataset of the object is saved down to a excel spreadsheet with the name of the dataset (Taken from the 'dataNames' property)

Input:

- obj : An object of class nb_bd

Optional input (..., 'propertyName', propertyName, ...):

- 'saveName' : A string with the wanted name of the saved file.

- 'ext' : > 'xlsx' : Saves the object down to a excel spreadsheet. Default

> 'matold' : Saves the data down to a mat file. The data is transformed into a structure, with the variables as the fieldnames, and the data as its fields. If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method with input old set to 1.

Must be combined with the optional input 'saveName'.

> 'mat' : Saves the data down to a mat file. The data is transformed into a structure, with the variables stored in the field 'variables', while the data of the variables will be saved as its fields with generic names ('Var1', 'Var2' etc). If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method.

Must be combined with the optional input 'saveName'.

> 'mats' : The object is saved down as a struct using the nb_ts.struct function.

> 'txt' : Saves the data down to a txt file.

> 'fame' ('db') : Writes a FAME database with the data stored in the nb_ts object. It is not possible to write to a database stored in the data warehouse.

- 'append' : > 1 : Saves all the datasets (pages of the data) to one excel spreadsheet, but in seperate worksheets. (Where the names of the worksheets are given by the dataNames

property of the object.)

Works only when 'ext' input is set to 'xlsx'. (The default)

> 0 : The default saving method.

- 'dateformat' : Sets the date format of the saved output Only if 'ext' are set to 'xlsx', which is default.
The supported date formats are:

> 'default'
 > 'fame'
 > 'xls'
 > 'NBNorsk' ('NBNorwegian')
 > 'NBEnglish' ('NBEngelsk')

See the toDates methods of the classes nb_day, nb_month, nb_quarter, nb_semiAnnual and nb_year for more information on the date formats

Output:

The nb_ts object saved to the wanted file format.

Examples:

Save the data to excel with the name 'test':
`saveDataBase(obj,'saveName','test');`

Save the data to excel with another date format:
`saveDataBase(obj,'saveName','test','dateformat','xls');`

Save the data to excel spreadsheet with the datasets as different worksheets:
`saveDataBase(obj,'saveName','test','append',1);`

Save the data to a .mat with the name 'test':
`obj.sav DataBase('ext','mat','saveName','test')`

See also:

[asCell](#), [toStructure](#), [nb_readMat](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

► **sec ↑**

`obj = sec(obj)`

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **secd** ↑

```
obj = secd(obj)
```

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **setInfinite2NaN** ↑

```
obj = setInfinite2NaN(obj)
```

Description:

Set all elements that are isfinite to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setMethodCalls** ↑

```
obj = setMethodCalls(obj,c)
```

Description:

Set all method calls of the object with a cell matrix. The object needs to be updatable.

Caution: To delete method calls use deleteMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.
- c : A cell matrix. See the output of getMethodCalls

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

See also:

[nb_bd.deleteMethodCalls](#), [nb_bd.getMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setValue ↑**

```
obj = setValue(obj,varname,value,startDateOfValues, ...
               endDateOfValues, pages)
```

Description:

Set some values of the dataset(s). (Only one variable at a time.)
You cannot use dates outside the objects dates (i.e. outside
obj.startDate:obj.endDate)

Input:

- obj : An object of class nb_bd
- varname : The variable name of the variable you want to set some values of. As a string
- value : The values you want to assign to the variable. Must be a numerical vector (with the same number of pages as given by pages or as the number of dataset the object consists of.) Cannot be outside the window of the data of the object.
- startDateOfValues : A date string with the start date of the data. If not given or given as a empty string it will assume that startDateOfValues to be the startDate of the object.

This input could also be an object which is of a subclass of the nb_date class.

- endDateOfValues : A date string with the end date of the data. If not given or given as a empty string it will assume that the endDateOfValues is the endDate of the object.

This input could also be an object which is

of a subclass of the nb_date class

- pages : At which pages you want to set the values of a variable. Must be a numerical index of the pages you want to set.
E.g. if you want to set the values of the 3 first datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages must be set.

Output:

- obj : An nb_bd object where the data of the given variable has been set.

Examples:

```
obj = nb_bd(ones(2,1),'', '2012', {'Var1'}, [zeros(2,1);1;],0)
```

```
obj = obj.setValue('Var1',[1;2;3]);  
obj = obj.setValue('Var1',2,'2013','2013',1);  
obj = obj.setValue('Var1',[1;2;3],'2012','2014');
```

See also
addVariable

Written by Per Bjarne Bye

► **setZero2NaN** ↑

```
obj = setZero2NaN(obj)
```

Description:

Set all elements that are 0 to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth SÅ!terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sin** ↑

```
obj = sin(obj)
```

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **sind** ↑

```
obj = sind(obj)
```

Description:

sind(obj) is the sine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

`sinh(obj)` is the hyperbolic sine of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = sinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **skewness ↑**

```
obj = skewness(obj,flag,outputType,dimension)
```

Description:

Calculate the skewness of each timeseries. The result will be an object of class `nb_bd` where all the non-nan values of all the timeseries are being set to their skewness.

The output can also be set to be a double or a `nb_cs` object consisting of the skewness values of the data.

Input:

- `obj` : An object of class `nb_bd`
- `flag` : Same as for the function `skewness` created by MATLAB.
- `outputType` :
 - > '`nb_bd`' : The result will be an object of class `nb_bd` where all the non-nan values of all the timeseries are being set to their skewness.
 - > '`double`' : Get the skewness values as doubles, where each column of the double matches the variable location in the '`variables`' property. If the object consist of more pages the double will also consist of more pages.

```

> 'nb_cs' : Get the skewness values as nb_cs object.
    The skewness will when this option is used
    only be calculated over the number of
    observations. (I.e. dimension set to 1.)

- dimension : The dimension to calcualate the skewness over.

Caution: the <ignorenan> property of obj controls the behavior of
        how to handle NaNs. This is equal to the 'notHandleNaN'
        optional input of the std method of the nb_ts class.

```

Output:

- obj : See the input outputType for more one the output from this method
- dimension : The dimension to calcualate the skewness over.

Examples:

```

nb_tsObj = skewness(obj);
double    = skewness(obj,0,'double');
nb_csObj = skewness(obj,0,'nb_cs');

```

Written by Per Bjarne Bye

► **sort ↑**

```
obj = sort(obj,dim,mode,variable)
```

Description:

Sort the nb_bd object in the given dimension

Caution : Sorting does not reorder the dataNames, variables or dates!

Input:

- obj : An object of class nb_bd
- dim : The dimension to sort
- mode : 'descend' (default)
'ascend'
- variable : Give a string with the variable to sort. The rest of the variables will then be reordered accordingly.

If empty (default), all variables are sorted.

Caution: Only an option for dim == 1

Output:

- obj : An nb_bd object representing the sorted input

Examples:

```
obj = nb_bd([1,2;1 1],'test','2012',{'Var1','Var2'});  
obj = sort(obj,1,'ascend')
```

Written by Per Bjarne Bye

► **sortProperty** ↑

```
obj = sortProperty(obj,type)
```

Description:

Sort the variables/types/datasets of the object alphabetically.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

sqrt(obj) is the square root of the elements of obj. Complex results are produced for non-positive elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **std ↑**

```
obj = std(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the std of each series. The result will be an object of class nb_bd where all the non-nan values of all the object are set to their std.

The output can also be set to be a double or a nb_cs object consisting of the skewness values of the data.

Input:

- obj : An object of class nb_cs

- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N

Where N is the sample length.

- outputType :

> 'double' : Get the std values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.

> 'nb_cs' : Get the std values as nb_cs object.

- dimension : The dimension to calcualate the std over.

Caution: the <ignorenan> property of obj controls the behavior of how to handle NaNs. This is equal to the 'notHandleNaN' optional input of the std method of the nb_ts class.

Output:

- obj : See the input outputType for available options.

Examples:

```
double    = std(obj,0,'double');
nb_csObj = std(obj,0,'nb_cs');
```

Written by Per Bjarne Bye

► **stopSorting** ↑

```
obj = stopSorting(obj)
```

Description:

Stop sorting of variables.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where sorting is turned off.

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **struct** ↑

```
s = struct(obj)
```

Description:

Object to struct.

Input:

- obj : An nb_bd object.

Output:

- s : A structure representing an nb_bd object.

Written by Per Bjarne Bye

► **structure2Dataset** ↑

```
obj = structure2Dataset(obj,givenStruct,NameOfDataset)
```

Description:

When you give a struct to the constructor of this class all the fields of that struct will be added as a new dataset, but this method add all the fields of the input givenStruct as a variable in one dataset

If you only want to add a dataset from a structure in this way (no other datasets) initialize a empty object, and then use this method.

Caution : The structure must include a field 'startDate', which must be a string with the start date of the data to add

Inputs:

- obj : The object itself, could be empty.
- givenStruct : The structure which contains the data of the added variable. Where the fieldnames will be the variables name of the added dataset + plus one field 'startDate' with the start date of the dataset.
- nameOfDataset : Name of the added dataset. If not provided default name will be used, i.e Database<jj>
- startDate : The start date of the new dataset. Only needed if the structure <givenStruct> has no field 'startDate'.

Outputs:

- obj : An nb_bd object with the added dataset.

Examples:

```
s          = struct();
s.startDate = '2012';
s.Var1      = [2;2;2];
obj        = nb_bd();
obj        = obj.structure2Dataset(s)

data =
    'Time'    'var1'
    '2012'    [ 2]
    '2013'    [ 2]
    '2014'    [ 2]
```

Written by Per Bjarne Bye

► **sum** ↑

```
obj = sum(obj,dim)
```

Description:

Takes the sum of the object timeseries

Caution : All sums ignore nan values no matter what the <ignorenan> property is set to (if not all values are nan).

Input:

- obj : An object of class nb_bd
- dim : The dimension to sum over, returns:
 - > dim = 1: Either :
 - > An nb_bd object with all the elements of each variable set to their sum over the time horizon.
 - > An nb_cs with one type representing the sum over the time horizon). The type is named 'sum'.
 - > dim = 2: An nb_bd object with one variable representing the sum (Take the sum over the variables)
(Default)
 - > dim = 3: An nb_ts object with only on page, representing the sum over all pages.
- output : Either 'nb_bd' (default) or 'nb_cs'. This is only important when summing over the 1 dimension.

Output:

- obj : Either an nb_bd object or a nb_cs object representing the sum.

Examples:

```
D    = {[1,2;3,4],[5,6;7,8]};  
Z    = cat(3,D{:});  
obj = nb_bd(Z,'','2012',{'Var1','Var2'})  
obj =  
  
(:,:,1) =  
  
'Time'    'Var1'    'Var2'  
'2012'    [ 1]    [ 2]  
'2013'    [ 3]    [ 4]
```

```

(:,:,2) =
    'Time'      'Var1'      'Var2'
    '2012'      [ 5]       [ 6]
    '2013'      [ 7]       [ 8]

s1 = sum(obj,1,'nb_bd')

s1 =
(:,:,1) =
    'Time'      'Var1'      'Var2'
    '2012'      [ 4]       [ 6]
    '2013'      [ 4]       [ 6]

(:,:,2) =
    'Time'      'Var1'      'Var2'
    '2012'      [ 12]      [ 14]
    '2013'      [ 12]      [ 14]

s1_cs = sum(obj,1,'nb_cs')

s1_cs =
(:,:,1) =
    'Types'      'Var1'      'Var2'
    'sum'        [ 4]       [ 6]

(:,:,1) =
    'Types'      'Var1'      'Var2'
    'sum'        [ 12]      [ 14]

s2 = sum(obj,2)

s2 =
(:,:,1) =
    'Time'      'sum'
    '2012'      [ 3]
    '2013'      [ 7]

(:,:,2) =
    'Time'      'sum'
    '2012'      [ 11]
    '2013'      [ 15]

s2 = sum(obj,3)

s2 =
    'Time'      'Var1'      'var2'
    '2012'      [ 6]       [ 8]

```

'2013' [10] [12]

Written by Per Bjarne Bye

► **tail** ↑

`tail(obj,nRows,page)`

Description:

Display the last n rows of a nb_bd object.

Input:

- `obj` : An nb_bd object.
- `nRows` : An integer with the number of rows to display. Defaults to 6.
- `page` : An integer with what page to display. Defaults to 1.

See also:

[nb_bd](#), [window](#), [head](#)

Written by Per Bjarne Bye

► **tan** ↑

`obj = tan(obj)`

Description:

Take the tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- `obj` : An object of class nb_ts, nb_cs or nb_data

Output:

- `obj` : An object of class nb_ts, nb_cs or nb_data

Examples:

`obj = tan(obj);`

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **tand** ↑

```
obj = tand(obj)
```

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

tanh(obj) is the hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **toStructure** ↑

```
S = toStructure(obj)
```

Description:

Transform the object to a structure. Data will be saved in its expanded form (i.e. filled with NaNs in place of missing values like nb_ts).

Input:

- obj : An object of class nb_bd

Output:

- S : A structure with fieldnames given by the object variables and variable data contained in the field.

Examples:

```
obj = nb_bd(ones(7,1),'',2012Q1,{'Var1'},[zeros(4,1);1;zeros(3,1)],0);
s = obj.toStructure()
s =
```

```
    Var1: [8x1 double]
    variables: {'Var1'}
    startDate: [1x1 nb_quarter]
    ignorenan: 1
    class: 'nb_bd'
    userData: ''
    localVariables: []
    sorted: 1
```

See also:

[nb_bd.struct](#)

Written by Per Bjarne Bye

► **tonb_ts** ↑

```
nb_ts_obj = tonb_ts(obj)
```

Description:

Transform from a nb_bd object to a nb_ts object

Input:

- obj : An object of class nb_bd

Output:

- nb_ts_obj : An object of class nb_ts

Examples:

nb_ts_obj = obj.tonb_ts();

Written by Per Bjarne Bye

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : A nb_ts, nb_cs or nb_data object where all the data are the unary minus of the input objects data

Examples:

obj = -obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **unstruct** ↑

obj = nb_bd.unstruct(s)

Description:

Reverse of the struct method.

Input:

- s : See the s output of the struct method.

Output:

- obj : An nb_bd object.

Written by Per Bjarne Bye

► **update ↑**

obj = update(obj,warningOff,inGUI)

Description:

This method will try to update the data of the nb_ts object.

Caution : It is only possible to update an nb_dataSource object which has updateable links to a FAME database or a full directory (path) to an excel spreadsheet or .mat file or the SMART database.

Caution : If you want to create a link to a specific worksheet of a excel file you must provide the extension!

Caution : All method calls done on the object are preserved.
(There exist some exception; and, or etc.)

Input:

- obj : An object of class nb_dataSource which are updatable.
- warningOff : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.

Output:

- obj : An object of class nb_dataSource with the data updated. If it is not possible a warning will be given.

Examples:

```
obj = nb_ts(['N:\Matlab\Utvikling\MATLAB_LIB\NEMO\'...
             'Documentation\Examples\example_ts_quarterly']);
obj = obj.update
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► uplus ↑

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► var ↑

```
obj = var(obj,outputType,dimension,varargin)
```

Description:

Calculate the var of object. The result will be an object of class nb_bd where all the non-nan values of all the object are being set to their variance.

The output can also be set to be a double or a nb_cs object consisting of the var values of the data.

Input:

- obj : An object of class nb_bd
- outputType :
 - > 'double' : Get the var values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.

```

> 'nb_cs' : Get the var values as nb_cs object.

- dimension : The dimension to calcualate the var over.

Caution: the <ignorenan> property of obj controls the behavior of
how to handle NaNs. This is equal to the 'notHandleNaN'
optional input of the std method of the nb_ts class.

```

Output:

- obj : See the input outputType for available options

Examples:

```

double    = var(obj,'double');
nb_csObj = var(obj,'nb_cs');

```

Written by Per Bjarne Bye

► **window ↑**

```
obj = window(obj,startDateWin,endDateWin,variablesWin,pages)
```

Description:

Narrow down window of the data of the nb_bd object

Input:

- obj : An object of class nb_bd
- startDateWin : The new wanted start date of the data of the object. Must be a string on the format; 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the start date of the data.
Can also be an object which is a subclass of the nb_date class.
- endDateWin : The new wanted end date of the data of the object. Must be a string on the format. 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the end date of the data.
Can also be an object which is a subclass of the nb_date class.
- variablesWin : A cellstr array of the variables you want to keep of the given object. If empty all the variables is kept.

- pages : A numerical index of the pages you want to keep.
E.g. if you want to keep the first 3 datasets
(pages of the data) of the object. And the
number of datasets of the object is larger than
3. You can use; 1:3. If empty all pages is kept.

I.e. `obj = obj.window('','',{},1:3)`

Output:

- obj : An nb_bd object where the datasets of the object is
narrowed down to the specified window.

Examples:

```
obj = nb_bd(ones(7,1),'','2012Q1',{'Var1'});  
obj = obj.window('2012Q2')
```

See also:

Written by Per Bjarne Bye

► **zeros** ↑

```
obj = nb_bd.zeros()  
obj = nb_bd.zeros(start,obs,vars,pages,sorted)
```

Description:

Create an nb_bd object with all data set to zeros.

Input:

- start : The start date of the created nb_bd object.
- obs : The number of observations of the created nb_bd object.
- vars : Either a cellstr with the variables of the nb_bd
object, or a scalar with the number of variables of the
nb_bd object.
- pages : Number of pages of the nb_bd object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_bd object.

Examples:

```
obj = nb_bd.zeros()
obj = nb_bd.zeros('2012Q1',10,['Var1','Var2','Var3']);
obj = nb_bd.zeros('2012Q1',2,2,10);
```

See also:

[nb_bd](#), [nb_bd.nan](#), [nb_bd.ones](#)

Written by Per Bjarne Bye

■ **nb_cell**

Go to: [Properties](#) | [Methods](#)

A class for storing data from cell (May be linked to an excel spreadsheet).

Constructor:

```
obj = nb_cell(datasets,NameOfDatasets)
obj = nb_cell(datasets,NameOfDatasets)
```

Input:

- datasets:

Input can be one of the data types listed below:

- xls(x) : A name of the excel spreadsheet to import. (With or without extension.). As a string.

Caution : If you read a specific worksheet, and want the object to be updateable you must provide the extension.

- cell : As a cell matrix. Each page of the matrix will be added as separate datasets.

You can also give a cellstr array consisting of more excel file names. Each excel file will be added as a new dataset.

- NameOfDatasets :

Input will depend on the data type provided to the datasets input:

- xls(x) : A string with the wanted dataset name. Default is the excel file name. If the provided string is a sheet of the read spreadsheet that specific worksheet will be read.

- cell : A string with the wanted dataset name. Default is 'Dataset1'.

Caution: When you given cell matrices which has more pages then one. Each of the pages will be added as a dataset.

The name of this datasets will, if the input NameOfDatasets has the same size as the input datasets, get the name NameOfDatasets{jj}, where jj is the page number, or else get the name 'Database<jj>'.

- A cellstr array of excel spreadsheet:

If you give a cellstr array to the datasets input, this input must be a cell array of strings. If you give a cellstr array which doesn't have the same size as the cellstr array given to the datasets input, the datasets which is left gets the name 'Database<jj>', where <jj> stands for this added datasets number.

If you give an empty cell array, i.e. {} all the datasets get the names given by a cellstr with the excel file names.

Output:

- obj : An object of class nb_cell.

Examples:

- xls(x) :

One excel spreadsheet

```
obj = nb_cell('test1')
```

More excel spreadsheets

```
obj = nb_cell({'test1','test2',...})
```

same as

```
obj = nb_cell({'test1','test2',...},{'test1','test2',...})
```

- cell matrix:

One matrix

```
obj = nb_cell(cell1)
```

```
obj = nb_cell(cell1,'test1')
```

One matrix with more pages:

```
obj = nb_cell(cell1,{})
```

```
obj = nb_cell(cell1,{'name1','name2',...})
```

See also:

[nb_table_cell](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- **cdata**
- **data**
- **dataNames**
- **localVariables**
- **numberOfDatasets**
- **numberOfVariables**
- **userData**
- **variables**

- **cdata** ↑

The data of the object as cell

- **data** ↑

The data of the object. As a double or logical.

Inherited from superclass NB_DATASOURCE

- **dataNames** ↑

The names of the different dataset. As a cellstr.

Inherited from superclass NB_DATASOURCE

- **localVariables** ↑

A nb_struct with the local variables of the nb_ts object.
I.e. a field with name 'test' can be reach with the string
input %#test. Only for dates and vintage inputs.

Inherited from superclass NB_DATASOURCE

- **numberOfDatasets** ↑

Number of datasets stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **numberOfVariables** ↑

Number of variables stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **userData** ↑

Add user data. Can be of any type.

Inherited from superclass NB_DATASOURCE

- **variables** ↑

Variables of the object. As a cellstr.

Inherited from superclass NB_DATASOURCE

Methods:

- abs
- acosh
- acoth
- acsch
- addDatasets
- addPostfix
- and
- asCellForMPRReport
- asech
- asinh
- atand
- callfun
- ceil
- cosd
- cotd
- csed
- cumnansum
- deleteMethodCalls
- deptcat
- empty
- expm1
- ge
- getCode
- getSource
- hasVariable
- horzcat
- isBoolean
- acos
- acot
- acsc
- addColumn
- addPageCopies
- addPrefix
- appendFromAnotherPage
- asec
- asin
- assignNan
- atanh
- callop
- conj
- cosh
- coth
- csch
- cumprod
- deleteVariables
- disp
- eq
- extMethod
- get
- getCreatedVariables
- getSourceList
- head
- interpolate
- isCrossSectional
- acosd
- acotd
- acscd
- addDataset
- addPages
- addRow
- asCell
- asecd
- asind
- atan
- breakLink
- callstat
- cos
- cot
- csc
- ctranspose
- cumsum
- demean
- double
- exp
- floor
- getClassOfData
- getMethodCalls
- gt
- histogram
- intertwine
- isDistribution

- [isDouble](#)
 - [isempty](#)
 - [isscalar](#)
 - [log](#)
 - [log2](#)
 - [merge](#)
 - [ne](#)
 - [ones](#)
 - [permute](#)
 - [rand](#)
 - [realsqrt](#)
 - [reorder](#)
 - [sec](#)
 - [setInfinite2NaN](#)
 - [setZero2NaN](#)
 - [sinh](#)
 - [stopSorting](#)
 - [subsref](#)
 - [tand](#)
 - [toStructure](#)
 - [unstruct](#)
 - [vertcat](#)
 - [zeros](#)
 - [isTimeseries](#)
 - [isfinite](#)
 - [keepPages](#)
 - [log10](#)
 - [lt](#)
 - [not](#)
 - [or](#)
 - [plot](#)
 - [real](#)
 - [rename](#)
 - [round](#)
 - [secd](#)
 - [setMethodCalls](#)
 - [sin](#)
 - [sortProperty](#)
 - [struct](#)
 - [tail](#)
 - [tanh](#)
 - [transpose](#)
 - [update](#)
 - [window](#)
 - [isUpdateable](#)
 - [isnan](#)
 - [le](#)
 - [log1p](#)
 - [map2Distribution](#)
 - [nan](#)
 - [objSize](#)
 - [packing](#)
 - [pow2](#)
 - [reallog](#)
 - [renameMore](#)
 - [saveDataBase](#)
 - [sech](#)
 - [setValue](#)
 - [sind](#)
 - [sqrt](#)
 - [subsasgn](#)
 - [tan](#)
 - [toMFile](#)
 - [uminus](#)
 - [uplus](#)
 - [writeTex](#)
-

► **abs ↑**

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = abs(in);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acos** ↑

obj = acos(obj)

Description:

Take acos of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acos(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acosd** ↑

obj = acosd(obj)

Description:

acosd(obj) is the inverse cosine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acosh** ↑

obj = acosh(obj)

Description:

acosh(obj) is the inverse hyperbolic cosine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acot** ↑

obj = acot(obj)

Description:

Take acotangent of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = acot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

acotd(obj) is the inverse cotangent, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acoth** ↑

```
obj = acoth(obj)
```

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acoth(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsc** ↑

obj = acsc(obj)

Description:

Take inverse csc of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acsc(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acscl** ↑

obj = acscl(obj)

Description:

acscl(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acscd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsch** ↑

```
obj = acsch(obj)
```

Description:

acsch(obj) is the inverse hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **addColumn** ↑

```
obj = addColumn(obj,location,elements)
```

Description:

Add a coulmn at a given location.

Input:

- obj : A nb_cell object.

- location : The location to add a column. It will be added after this column. If empty a empty row will be added last in the nb_cell object.

- elements : A cell with size size(obj.cdata,1) x 1 x size(obj.cdata,3). If empty, a empty column will be added.

Output:

- obj : A nb_cell object with a added column.

See also:

[addRow](#)

Written by Kenneth Sæterhagen Paulsen

► **addDataset** ↑

```
obj = addDataset(obj,dataset,NameOfDataset)
```

Description:

Makes it possible to add one dataset to a existing nb_cell object

Caution : If more than one dataset is added to an object the link to the data source is broken!

Input:

- obj : An object of class nb_cell
- dataset : Either a string with the name of excel spreadsheet or a cell matrix.
- NameOfDataset : Must be a string with the dataset name. If not given (or given as '') the default name Database<jj> is given. Where jj is the number of added datasets of the object, including dataset you add with this method.

Output:

- obj : The object itself with the added dataset.

Examples:

```
obj = nb_cell;
obj = obj.addDataset([2,2],'test',{'First'},{'Var1','Var2'});
```

Written by Kenneth S. Paulsen

► **addDatasets** ↑

```
obj = addDatasets(datasets,NameOfDatasets)
```

Makes it possible to add more datasets to a existing nb_cell object

Input:

Same input as of the constructor but only cell arrays are supported for the inputs 'datasets' and 'NameOfDatasets'. 'NameOfDatasets' could be given as a empty cell; {};

Output:

- obj : An object of class nb_cell, now including the new datasets provided

Examples:

```
obj = addDatasets({'excel1','excel2'},{},{'First'})
```

Written by Kenneth S. Paulsen

► **addPageCopies** ↑

Description:

Add copies of the data of an object of class nb_cell and append it as new datasets of the object. Only possible if an object has only one page.

Input:

- obj : An object of class nb_cell
- num : Number of copies to be appended

Output:

- obj : An object of class nb_cell with the added copies of the first dataset in the object.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = addPageCopies(obj,2);
```

Written by Kenneth S. Paulsen

► **addPages** ↑

```
obj = addPages(obj,DB)
```

Description:

Add all pages from a different nb_cell object with the current one

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_cell
- DB : An object of class nb_cell
- varargin : Optional number of objects of class nb_cell

Output:

- obj : An object of class nb_cell with all the datasets from the objects obj and varargin.

Be aware: Objects will be expanded with nans to match sizes.

Examples:

```
obj = addPages(obj,DB);
```

Written by Kenneth S. Paulsen

► **addPostfix ↑**

```
obj = addPostfix(obj,postfix)
```

Description:

Add a postfix to all elements of the data of the nb_cell object.

Input:

- obj : An object of class nb_cell
- postfix : The prefix to add to all string elements of the objects cell data.

Output:

- obj : An object of class nb_cell with the postfix added.

Examples:

```
obj = obj.addPostfix('_NEW');
```

Written by Kenneth S. Paulsen

► **addPrefix** ↑

```
obj = addPrefix(obj,prefix)
```

Description:

Add a prefix to all elements of the data of the nb_cell object.

Input:

- obj : An object of class nb_cell
- prefix : The prefix to add to all string elements of the objects cell data.

Output:

- obj : An object of class nb_cell with the prefix added.

Examples:

```
obj = obj.addPrefix('_NEW');
```

Written by Kenneth S. Paulsen

► **addRow** ↑

```
obj = addRow(obj,location,elements)
```

Description:

Add a row at a given location.

Input:

- obj : A nb_cell object.
- location : The location to add a row. It will be added after this row.
If empty a empty row will be added last in the nb_cell object.
- elements : A cell with size 1 x size(obj.cdata,2) x size(obj.cdata,3).
If empty, a empty row will be added.

Output:

- obj : A nb_cell object with a added row.

See also:

[addColumn](#)

Written by Kenneth Sætherhagen Paulsen

► **and** ↑

a = and(a,b)

Description:

The and operator (&).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the and operator has evaluated all the data elements of the object.
The data will be a logical matrix

Examples:

a = a & b;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **appendFromAnotherPage** ↑

obj = appendFromAnotherPage(obj,page)

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_dataSource object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_dataSource object.

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_DATASOURCE

► asCell ↑

```
cellMatrix = asCell(obj,page)
```

Description:

Return the nb_cell objects data as a cell matrix.

Input:

- obj : An object of class nb_cell

- page : Which pages should be transformed to a cell. Must be a double (vector) with the indices.

Output:

- cellMatrix : The objects data transformed to a MATLAB cell.

Written by Kenneth S. Paulsen

► asCellForMPRReport ↑

```
cellMatrix = asCellForMPRReport(obj,roundoff)
```

Description:

Return the nb_cell objects data as a cell matrix, on the MPR publishing format

Input:

- obj : An object of class nb_cell

- roundoff : 1 if you want to round off to 2 decimals, 0 otherwise. 1 is default.

Output:

- `cellMatrix` : The objects data transformed to a cell. Only the first page is returned. Look very much like an excel spreadsheet.

Written by Kenneth S. Paulsen

► **asec** ↑

`obj = asec(obj)`

Description:

Take inverse sec of the data stored in the `nb_ts`, `nb_cs` or `nb_data` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`.

Examples:

`obj = asec(obj);`

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **asecd** ↑

`obj = asecd(obj)`

Description:

`asecd(obj)` is the inverse secant, expressed in degrees, of the elements of `obj`

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = asecd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE
```

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asech(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE
```

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sin of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = asin(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_DATASOURCE
```

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asinh(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► assignNan ↑

```
obj = expand(obj,variables,value)
```

Description:

Assign all the nan observation to the value input.

Input:

- obj : An object of class nb_dataSource.
- variables : A char or a cellstr with the variables to assign nan values.
- value : A scalar number.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj = nb_ts.rand('2012Q1',10,2,3);  
obj(1:2,1,:) = nan;  
obj = assignNan(obj,'Var1',0)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► atan ↑

```
obj = atan(obj)
```

Description:

Take the inverse tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees, of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► breakLink ↑

```
obj = breakLink(obj)
```

Description:

Break link to source

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► callfun ↑

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s). The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb.DataSource.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nb.DataSource.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb_dataSource objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb_dataSource object. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d      = nb_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)
```

See also:

[nb_cell.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callop ↑**

```
obj = callop(obj,another,func)
obj = callop(obj,another,func,type)
```

Description:

Call mathematical operator on the data of the object. In contrast to calling the operator itself on the object, this method does not try to match the same variables from the two inputs, but instead operate in the same way as mathematical operators do on double matrices (bsxfun). The time domain is matched though!

Caution: The following operators are supported:

- > @plus or 'plus'
- > @minus or 'minus'
- > @times or 'times'
- > @rdivide or 'rdivide'
- > @power or 'power'

Input:

- obj : An object of class nb_dataSource.
- another : An object of class nb_dataSource.
- func : One of the above listed functions.
- type : Either:
 - > 'keep' : Keep the variable names of the object with most variables. If they have the same number of variables, they are taken from the first object.
 - > 'rename' : The new variable names represent the operation that has been done to the separate series. So if you divide an object with one series named 'Var1' with another object with one series named 'Var2', the new name will be 'Var1./Var2'. (Default)

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb_dataSource method. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});  
d2 = nb_ts(rand(10,1),'''2011Q1',{'Var2'});  
d3 = callop(d1,d2,@minus);  
  
d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});  
d2 = nb_ts(rand(10,2),'''2011Q1',{'Var2','Var3'});  
d3 = callop(d1,d2,@minus);
```

See also:

[nb_cell.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callstat ↑**

```
obj = callstat(obj,func)  
obj = callstat(obj,func,varargin)
```

Description:

Call a in-built or user defined function on the data of the nb_dataSource object(s) that goes from the data of the object have more than one variable to only having one. E.g. when taking the mean over a set of variables.

Input:

- obj : An object of class nbDataSource.
- func : A function handle (or one line char) that maps a nObs x nVar x nPage double to nObs x 1 x nPage double, or that maps a nObs x nVar x nPage double to nObs x nVar x 1 double.

Optional input:

- 'name' : Set the name of the new variable, as a one line char. Default is to use the str2func on the provided function handle.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- obj : An object of class nbDataSource with only one variable.

Examples:

```
d = nb_ts.rand('2012Q1',10,3);
d1 = callstat(d,@(x)mean(x,2),'name','mean')
```

See also:

[nb_cell.mean](#), [nb_cell.std](#), [nb_cell.var](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ceil ↑**

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = ceil(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **conj** ↑

obj = conj(obj)

Description:

conj(obj) is the complex conjugate of the elements of obj.
For a complex element x of obj, conj(x) = REAL(x) - i*IMAG(x).

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = conj(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cos** ↑

obj = cos(obj)

Description:

Take cos of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

cosd(obj) is the cosine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cosd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cosh** ↑

```
obj = csch(obj)
```

Description:

cosh(obj) is the hyperbolic cosine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = csch(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cot** ↑

obj = cot(obj)

Description:

Take cotangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = cot(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cotd** ↑

obj = cotd(obj)

Description:

cotd(obj) is the cotangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = cotd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **coth** ↑

obj = coth(obj)

Description:

coth(obj) is the hyperbolic cotangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = coth(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csc** ↑

obj = csc(obj)

Description:

Take csc of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = csc(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cscd** ↑

obj = cscd(obj)

Description:

cscd(obj) is the cosecant of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = cscd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csch** ↑

obj = csch(obj)

Description:

csch(obj) is the hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **ctranspose** ↑

```
obj = ctranspose(obj)
```

Description:

Take the transpose ('') of an nb_cell object

Caution : The former types will be sorted when transposing the object.

Input:

- obj : An object of class nb_cell

Output:

- obj : An nb_cell object which represents the transposed input object

Written by Kenneth S. Paulsen

► **cumnansum** ↑

```
obj = cumnansum(obj)
```

Description:

Cumulativ sum of the series of the object. Ignoring nan values.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumnansum(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumprod** ↑

```
obj = cumprod(obj,dim)
```

Description:

Cumulativ product of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : A nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative product of the old objects 'data' property.

Examples:

```
obj = cumprod(obj);
obj = cumprod(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumsum(obj);
obj = cumsum(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteMethodCalls** ↑

```
obj = deleteMethodCalls(obj,c)
```

Description:

Delete some method calls of the object. The object needs to be updatable.

Caution: To set method calls use setMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- sourceNr : The source index to delete the methods from
- methodNrs : A index with the methods to delete.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

See also:

[setMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteVariables** ↑

```
obj = deleteVariables(obj,deletedVar)
```

Description:

Delete variables from the current nb_ts, nb_data or nb_cs object (letter size has something to say)

If some of the variables you specify in deletedVar does not exist in the obj, this will not have anything to say for the obj. (The variables cannot be deleted, because the variables do not exist.)

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- deletedVar : Must be a char or a cellstr

Delete all the variable names, given in the char array or cellstr array deletedVar, of current the object.

NB! If deletedVar is just one string and include a * as the last letter, this function will delete all the variable names, from the object, that start with the letters which is in front of the * in the string.

i.e. `obj = deleteVariables(obj,'DPQ*');`

Deletes all the variables in the obj which has variable names starting with 'DPQ'.

If deletedVar is just one string and include a * as the first letter, this function will delete all the variable names, from the object, that include the letters which follows in the string.

i.e. `obj = deleteVariables(obj, '*shift');`

Deletes all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_ts, nb_data or nb_cs object with the variables specified in deletedVar deleted.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = deleteVariables(obj, {'Var1','Var2'});  
obj = deleteVariables(obj, char('Var1','Var2'));  
obj = deleteVariables(obj, obj.variables);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_dataSource object.
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_dataSource object.

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATASOURCE

► **deptcat** ↑

```
obj = deptcat(obj,varargin)
```

Description:

Depth concatenation (add pages of different objects) (No short notation)

Be aware: The sizes of the cell matrices will be expanded automatically to match eachother.

Input:

- obj : An object of class nb_cell
- varargin : Optional number of nb_cell objects

Output:

- obj : An object of class nb_cell where all the objects datasets are added into.

See also:

[addPages](#)

► **disp ↑**

`disp(obj)`

Description:

Display the object on the command line (Without semicolon)

Input:

- `obj` : An object of class nb_cell

Output:

The nb_cell object displayed on the command line if you let out the semicolon at the end of the code line

Written by Kenneth S. Paulsen

► **double ↑**

`dataOfObject = double(obj)`

Description:

Get the data of the object as a double matrix

Caution : If the data of the object is represents as an object of class nb_distribution the mean is returned.

Input:

- `obj` : An object of class nb_ts, nb_cs or nb_data

Output:

- `dataOfObject` : The data of the nb_ts, nb_cs or nb_data object

Examples:

`dataOfObject = double(obj)`

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **empty** ↑

```
obj = empty(obj)
```

Description:

Empty the nb_cs object

Input:

- obj : An object of class nb_cs

Output:

- obj : An empty nb_cs object

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = empty(obj);
```

Written by Kenneth S. Paulsen

► **eq** ↑

```
a = eq(a,b)
```

Description:

Test if the one object is equal (==) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **exp ↑**

```
obj = exp(obj)
```

Description:

Take exp of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **expm1 ↑**

```
obj = expm1(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, e to the `obj`.
`expm1(obj)` computes `exp(obj)-1`, compensating for the roundoff in
`exp(obj)`.

(For small real X, `expm1(X)` should be approximately X, whereas the
computed value of EXP(X)-1 can be zero or have high relative error.)

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are equal to e.^obj.data-1

Examples:

```
obj = expm1(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► extMethod ↑

```
obj = extMethod(obj,method,variables,postfix,varargin)
```

Description:

This method can call a nb_dataSource method on a selected variables, and merge the result with the old object by adding a postfix to the new variables.

This method is the same as calling:

```
obj1 = window(obj,'','variables');
method = str2func(method);
obj = method(obj,varargin{:});
obj1 = addPostfix(obj1,method);
obj = merge(obj,obj1);
```

But without replicating the links property unnecessary many times!

Caution : If the object is not updateable this will result in the same as the code above!

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- method : A str with the method to call. Must be a method of the nb_ts, nb_data or nb_cs class that does not change other dimensions then the second dimension!
- variables : A cellstr with the variables of the object to call the method on. Must be included in the object.
- postfix : A string with the postfix. If empty, the old variables will be replaced by the new ones.

Optional inputs:

These will be the inputs casted to the method except the object itself.

Extra for nb_ts and nb_data:

- '<start>' : The start date/obs of the function evaluation. Given as the second input to the window method call.
- '<end>' : The end date/obs of the function evaluation. Given as the third input to the window method call.

Output:

- obj : A nb_ts, nb_data or nb_cs object

Examples:

```
obj = nb_ts.rand(1,10,3);
obj = extMethod(obj,'lag',{'Var1'},'lag1',1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **floor** ↑

```
obj = floor(obj)
```

Description:

floor(obj) rounds the data elements of obj to the nearest integers towards minus infinity

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = floor(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **ge** ↑

```
a = ge(a,b)
```

Description:

Test if the one object is greater than or equal to (\geq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get the properties of the object

Input:

- obj : An object of class nb_ts
- propertyName : Name of the property you want to get:
 - 'cdata' : The data of the object as a cell.
 - 'data' : The data of the object as a double
 - 'dataNames' : Name of the object datasets, as cellstr array
 - 'numberOfDatasets' : Number of datasets (pages) of the object. Size of the third dimension
 - 'links' : A structure of the data source links.
 - 'updateable' : 1 if updateable, 0 otherwise.

Output:

```
propertyValue : The property value of the object for the wanted  
                  property
```

Examples:

```
dataAsDouble = get(obj,'data');  
variables    = get(obj,'variables')
```

Written by Kenneth S. Paulsen

► getClassOfData ↑

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_dataSource.

Output:

- cl : Name of the class that the data of the object is stored as.

See also:

[nb_cell.isDistribution](#), [nb_cell.isDouble](#)
[nb_cell.isBoolean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getCode ↑

```
code = getCode(obj,preamble)  
code = getCode(obj,preamble,'inputName',inputValue,...)
```

Description:

Get the LaTeX code of a table representing the data of the object

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_cell
- preamble : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.

- 'fontSize' : Sets the font size of the table. Either:
'Huge', 'huge', 'LARGE', 'Large', 'large',
'normalsize' (default), 'small',
'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.

- 'justification' : Sets the justification of the caption of the table. Must be a string.
Either 'justified','centering' (default),
'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.

- language : Either 'english' (default) or 'norwegian'
('norsk').

- lookUpMatrix :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';  
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
lookUpMatrix = {  
  
'mnemonics1','englishDescription1','norwegianDescription1';  
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined char, which will result in multi-line text of the table.

Caution : This will act on both variables and types.

- 'precision' : Sets the precision of the numbers in the table. Must be a string. Default is '%3.2f'. I.e. two decimals.
- 'rotation' : The rotation of the table in LaTeX. Either:
 - > 'sidewaystable' : A sideways table
 - > 'landscape' : Rotate the whole page
 - > '' : No rotation
- 'rowColor1' : Color of every second row of the table starting from the second row. Same options as was the case for the 'firstRowColor' option.
- 'rowColor2' : Color of every second row of the table starting from the third row. Same options as was the case for the 'firstRowColor' option.

Output:

- code : A char with the tex code of the table

Examples:

```
obj = nb_cell({'Test', 2, 'Test2', 3});
code = obj.getCode(1)
```

See also:

[nb_cell](#)

Written by Kenneth Sæterhagen Paulsen

► **getCreatedVariables ↑**

```
created = getCreatedVariables(obj)
```

Description:

Get a N x 2 table of the variables created using the createVariable method. The first column list the created variables, while the second column list the expressions.

Input:

- obj : An object that is a subclass of nb_dataSource.

Output:

- created : A N x 2 cell array. See description of this method for more.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getMethodCalls** ↑

```
[s,c] = getMethodCalls(obj)
```

Description:

Get all method calls as a cell matrix. The object needs to be updatable to get a list of methods called on the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

- c : A cell matrix with a table of the method called on the object and its input. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSource** ↑

```
sourceType = getSource(obj)
sourceType = getSource(obj,type)
```

Description:

Get source type.

Input:

- obj : An object of class nb_dataSource.

- type : Give 'program' to get the general source. Default is to give the specific type of source.

Output:

- sourceType : A one line char with the source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSourceList** ↑

s = getSourceList(obj)

Description:

Get the source list of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **gt** ↑

a = gt(a,b)

Description:

Test if the one object is greater then (>) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.

- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a > b;  
obj = 2 > b;  
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hasVariable** ↑

```
[ret,location] = hasVariable(obj,variable)
```

Description:

Test if an nb_cs object has a given variable

Input:

- obj : An object of class nb_cs
- variable : The variable name as a string. Can also be a cellstr with more variables.

Output:

- ret : 1 if found otherwise 0
- location : The location of the variable if found. Otherwise [].

Examples:

```
ret = obj.hasVariable('Var1')  
[ret,location] = hasVariable(obj,'Var1')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **head** ↑

```
head(obj,nRows,page)
```

Description:

Display the first n rows of a nb_cell object.

Input:

- obj : An nb_cell object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_cell](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **histogram** ↑

```
data          = histogram(obj)
[data,plotter] = histogram(obj,M)
```

Description:

Create a histogram of a object containing only one variable and one page!

Input:

- obj : A nb_dataSource object with size N x 1 x 1.
- M : The number of bins of the histogram.

Output:

- data : A nb_cs object with size M x 1 x 1.
- plotter : A nb_graph_cs object with the histogram plotted.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **horzcat** ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation of nb_cell objects. ([a,b])

Input:

- a : An object of class nb_cell
- varargin : Optional number of nb_cell objects

Output:

- obj : An nb_cell object with all the variables from the different nb_cell object merge into one dataset

Examples:

```
obj = [a,b];
obj = [a,b,c];
```

See also:

[merge](#)

Written by Kenneth S. Paulsen

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_data object. Will discard leading and trailing nan values.

Input:

- data : An nb_ts, nb_cs or nb_data object.
- method :
 - 'nearest' : Nearest neighbor interpolation
 - 'linear' : Linear interpolation. Default
 - 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - 'pchip' : Shape-preserving piecewise cubic interpolation
 - 'cubic' : Same as 'pchip'
 - 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : An nb_ts, nb_cs or nb_data object with the interpolated data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **intertwine** ↑

```
obj = intertwine(varargin)
```

Description:

Intertwine objects of class nb_cell objects with equal sizes.

Caution: This method will break the link to the data source.

Optional input:

- 'header' : Intertwine header, i.e. first row. true or false.
Default is false.
- varargin : Supply nb_cell objects (obj, obj2, ...)

Output:

- obj : An object of class nb_cell

Written by Kenneth Sæterhagen Paulsen

► **isBoolean** ↑

```
ret = isBoolean(obj)
```

Description:

Check if the data of the nb_dataSource object is of class logical (boolean, i.e. true or false).

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isCrossSectional** ↑

```
ret = isCrossSectional(obj)
```

Description:

Test if this object is a cross sectional data object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_cs, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isDistribution** ↑

```
ret = isDistribution(obj)
```

Description:

Check if the data of the nb_dataSource object is of class nb_distribution

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isDouble** ↑

```
ret = isDouble(obj)
```

Description:

Check if the data of the nb_dataSource object is of class double (numeric)

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isTimeseries** ↑

```
ret = isTimeseries(obj)
```

Description:

Test if a nb_dataSource object is a time series object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_ts, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isUpdateable** ↑

```
ret = isUpdateable(obj)
```

Description:

Test if this object is updateable.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : 1 if updateable, otherwise 0.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Test if each element of a nb_ts, nb_cs or nb_data object is finite.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE
```

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_ts, nb_cs or nb_data object is nan.
Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE
```

► **isscalar** ↑

```
ret = isscalar(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is a scalar. Will always return 0, as an object of class nb_ts, nb_cs or nb_data is not a scalar.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : false

Examples:

```
0 = isscalar(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepPages ↑**

```
obj = keepPages(obj, pages)
```

Description:

Keep pages of the current nb_ts object

Input:

- obj : An object of class nb_ts
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty a empty nb_ts object is returned.

Can also be a cellstr with the names to keep, or a logical array with length equal to the number of pages (datasets).

Output:

- obj : An nb_ts object with the pages specified in the input are kept.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **le ↑**

```
a = le(a,b)
```

Description:

Test if the one object is less than or equal to (\leq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log ↑**

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on logs.

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log10 ↑**

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on (base 10) logs.

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► log1p ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► log2 ↑

```
obj = log2(obj)
```

Description:

`log2(obj)` is the base 2 logarithm of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **lt** ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less than (`<`) the other object elemetswise and return a `nb_dataSource` object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- `a` : An object of class `nb_dataSource` or a scalar number.

- `b` : An object of class `nb_dataSource`, but must be of same subclass as `a`, or a scalar number.

Output:

- `a` : An `nb_dataSource` object where the data element are logical.

Examples:

```
obj = a < b;
obj = 2 < b;
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **map2Distribution** ↑

```
obj = map2Distribution(obj,dist)
obj = map2Distribution(obj,dist,varargin)
```

Description:

Map observation to the distributions given by dist.

The current distribution of the data is estimated using a kernel density estimator. The data must be strongly stationary. I.e. the unknown distribution from where the data has been drawn must not depend on time.

Input:

- obj : An object of class nb_dataSource.
- dist : An object of class nb_distribution with size 1 x size(obj,2).

Optional inputs:

- See the optional inputs of the nb_distribution.estimate function.

Output:

- obj : An object of class nb_dataSource.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **merge** ↑

```
obj = merge(obj,DB,'horzcat')
obj = merge(obj,DB,'vertcat')
```

Description:

Merge another nb_cell object with the current one

Input:

- obj : An object of class nb_cell
- DB : An object of class nb_cell
- type : Either 'horzcat' or 'vertcat'. Dimensions must agree!

Output:

- obj : An object of class nb_cell.

Written by Kenneth S. Paulsen

► **merge2Series ↑**

```
obj = merge2Series(obj,var1,var2,new,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_dataSource.
- var1 : A one line char with the name of the base series.
- var2 : A one line char with the name of the appended series.
- new : Name of the new variable. If given as empty there will not be created a new series, but instead the var1 variable will be set to the new merged series. Default is empty.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_dataSource.

Examples:

```
obj           = nb_ts.rand('2012Q1',10,2,3);
obj(end-1:end,1,:) = nan;

obj1 = merge2Series(obj,'Var1','Var2','Var3')
obj2 = merge2Series(obj,'Var1','Var2')
obj3 = merge2Series(obj,'Var1','Var2','','diff','nLags',1)
obj4 = merge2Series(obj,'Var1','Var2','','diff','nLags',4)
obj5 = merge2Series(obj,'Var1','Var2','','level','date','2014Q1')
```

See also:

[nb_ts.merge](#), [nb_data.merge](#), [nb_cs.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► [nan](#) ↑

```
obj = nb_cell.nan(rows,columns)
obj = nb_cell.nan(rows,columns,pages)
```

Description:

Create an nb_cell object with all elements set to nan.

Input:

- rows : Number of rows of the nb_cell object.
- vars : Number of columns of the nb_cell object.
- pages : Number of pages of the nb_cell object.

Output:

- obj : An nb_cell object.

See also:

[nb_cell](#)

Written by Kenneth Sæterhagen Paulsen

► [ne](#) ↑

```
a = ne(a,b)
```

Description:

Test if the one object is not equal to ($\sim=$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **not ↑**

```
a = not(a)
```

Description:

The not operator (~) .

Input:

- a : An object of class nb_dataSource.

Output:

- a : An object of class nb_dataSource. Where the not operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = ~a;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **objSize ↑**

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim :
 - 1 : Number of observations
 - 2 : Number of variables
 - 3 : Number of datasets (pages)

Output:

- varargout : See the examples below

Examples:

```
dim = objSize(obj)
```

Where dim is 1 x 2 double where the first element is the number of observations/types and the second element is the number of variables

```
[dim1, dim2] = objSize(obj)
```

```
[dim1, dim2, dim3] = objSize(obj)
```

```
dim1 = objSize(obj,1)  
dim2 = objSize(obj,2)  
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **ones ↑**

```
obj = nb_cell.ones(rows,columns)  
obj = nb_cell.ones(rows,columns,pages)
```

Description:

Create an nb_cell object with all elements set to 1.

Input:

- rows : Number of rows of the nb_cell object.
- vars : Number of columns of the nb_cell object.
- pages : Number of pages of the nb_cell object.

Output:

- obj : An nb_cell object.

See also:

[nb_cell](#)

Written by Kenneth Sæterhagen Paulsen

► [or](#) ↑

a = or(a,b)

Description:

The and operator (|).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the or operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

a = a | b;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **packing** ↑

```
obj = packing(obj,packages,varargin)
```

Description:

Package variables by applying the wanted function. Default is summin (sum).

Input:

- obj : An object of class nb_ts.
- packages : A 2 x N cell matrix with groups of variables and the names of the groups. If you have not listed a variable it will be grouped in a group called 'Rest' (as long as 'includeRest' is not set to false).

Must be given in the following format:

```
{'group_name_1','group_name_N';  
 name_1 ,...,name_N}
```

Where name_x must be a string with a name of the variable or a cellstr array with the variable names. E.g 'Var1' or {'Var1','Var2'}.

Optional inputs:

- 'includeRest' : Give false to not include the 'Rest' variable, which will represent all the variable that has not been packed.
- 'func' : Either a function_handle or a one line char with the name of the function to use. The first input to this function is a nObs x nVar double, the second is the dimension (which is always 2). Examples; 'sum', 'prod', @sum or @prod.

Output:

- obj : A object of class nb_ts.

Examples:

```
data      = nb_ts.rand('2012Q1',10,4);  
packages = {'group1','group2';  
            {'Var1','Var2'},{'Var3'}};  
  
dataP1 = packing(data,packages)  
dataP2 = packing(data,packages,'includeRest',false)  
dataP3 = packing(data,packages,'func',@prod)
```

See also:

[nb_ts.createVariable](#), [nb_data.createVariable](#), [nb_cs.createVariable](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **permute** ↑

```
obj = permute(obj,variables)
```

Description:

Switch the second and third dimension of the nb_ts, nb_cs or nb_data object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- variables : A cellstr with size 1 x numberOfDatasets. Default is to use dataNames property. I.e. when not provided or if empty.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **plot** ↑

```
plot(obj,varargin)
plotter = plot(obj,varargin)
```

Description:

Plot the data of the nb_dataSource object.

Caution: If the data of the object is of class nb_distribution this method will redirect to plotDist.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- plotType : A string with either:
 - > 'graph' (Default)

```

> 'graphSubPlots'

> 'graphInfoStruct'

The name of the graphing method of the class
nb_graph_ts, nb_graph_cs or nb_graph_data.

Caution: This is an optional input. If not given it is
assumed that the optional input pairs are starting from the
second input to this function.

- varargin : Same as the input to the method set() of the
  nb_graph_ts, nb_graph_cs or nb_graph_data class.

I.e. ..., 'inputName', inputValue, ...

Caution: If the data of the object is of class
  nb_distribution, see nb_ts.plotDist for the
optional input pairs supported.

```

Output:

Plotted output

Examples:

```

plot(obj)

plot(obj,'graphSubPlots')

plot(obj,'','startGraph','2008Q1','endGraph','2011Q2')

same as

plot(obj,'graph','startGraph','2008Q1','endGraph','2011Q2')

```

See also:

[nb_graph_ts](#), [nb_ts.plotDist](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **pow2 ↑**

obj = pow2(obj)

Description:

pow2(obj) raises the number 2 to the power of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **rand** ↑

```
obj = nb_cell.rand(rows,columns)
obj = nb_cell.rand(rows,columns,pages)
obj = nb_cell.rand(rows,columns,pages,dist)
```

Description:

Create a random nb_cell object

Input:

- rows : Number of rows of the nb_cell object.
- vars : Number of columns of the nb_cell object.
- pages : Number of pages of the random nb_cell object.
- dist : A nb_distribution object to draw from.

Output:

- obj : An nb_cell object.

Examples:

```
obj = nb_cell.rand(2,2);
obj = nb_cell.rand(2,2,10,nb_distribution('type','normal'));
```

See also:

nb_cell

Written by Kenneth Sæterhagen Paulsen

► **real** ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part of the elements in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **reallog** ↑

```
obj = reallog(obj)
```

Description:

Take the real natural logarithm of the data stored in the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = reallog(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **realsqrt** ↑

```
obj = realsqrt(obj)
```

Description:

realsqrt(obj) is the square root of the elements of obj.
An error is produced if obj contains negative elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = realsqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **rename** ↑

```
obj = rename(obj,type,oldName,newName)
```

Description:

Makes it possible to rename variables and datasets. For objects of class nb_cs types is also possible to rename.

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variable', but not when the input is 'dataset' or 'types'.

Input:

- obj : An object of class nb_dataSource
- type : Either:
 - 'variable' : To rename a variable(s)
 - 'dataset' : To rename a dataset(s) (The only input that is supported for nb_cell)
 - 'types' : To rename a type(s) (only nb_cs)
- oldName : Case 1;

> The old variable/dataset/type name, as a string

Case 2:

> The string which should be replaced in all the variable names of the database. And reorder things

afterwards. This input must end with a *.

```
- newName : Case 1;  
    > The new variable/dataset/type name, as a string  
  
Case 2;  
    > The string which should replace the old string part  
      given by the input 'oldName'.
```

Output:

- obj : An nb_dataSource object with the renamed variable(s), type(s) or dataset(s).

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.rename('variable','Var1','Var3');  
obj = obj.rename('variable','Var*','N_Var');  
obj = rename(obj,'variable',{'Var1','Var2'},{'VAR1','VAR2'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **renameMore** ↑

```
obj = renameMore(obj,type,oldNames,newNames)
```

Description:

Makes it possible to rename more variables, datasets and types (only nb_cs).

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variables', but not when the input is 'datasets'.

Caution : This is the same as looping over nb_dataSource.rename, but more efficient.

Caution : * syntax is not supported as is the case for nb_dataSource.rename

Input:

```
- obj      : An object of class nb_dataSource.  
- type    : Either:  
            - 'variables' : To rename a variables  
            - 'datasets'   : To rename a datasets (The only  
                            input that is supported for nb_cell)  
            - 'types'      : To rename a type (Only nb_cs)
```

- oldNames : The old variable/dataset/type names, as a cellstr.
- newName : The new variable/dataset/type names, as a cellstr.

Output:

- obj : An nb_dataSource object where the variable(s)/dataname(s)/type(s) of the object given as input is/are renamed.

Examples:

```
obj = nb_cs([22,24;27,28],'Dataset1',{'Exp','Imp'},{'Nor','Swe'})
obj = renameMore(obj,'dataset',{'Dataset1'},{'tradeBal'});
obj = renameMore(obj,'variable',{'Nor','Swe'},{'Norway','Sweden'});
obj = renameMore(obj,'type',{'Exp','Imp'},{'Export','Import'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **reorder** ↑

```
obj = reorder(obj,newOrder,type)
```

Description:

Re-order the variables/types/datasets of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- newOrder : A cellstr with the new order of the variables/types/datasets.
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **round** ↑

```
obj = round(obj,value)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_cell object
- value : The rounding value. As a double. Default is 1.
- pages : A numerical index of the pages you want to keep.

Output:

- obj : A nb_cell object with the rounded numbers (only numerical values).

Written by Kenneth Sæterhagen Paulsen

► saveDataBase ↑

```
saveDataBase(obj,varargin)
```

Description:

Save data of the nb_cell object

If no optional input is given the data of the object is save down to xlsx file(s). Each dataset of the object is saved down to a excel spreadsheet with the name of the dataset (Taken from the dataNames property of the object)

Input:

- obj : An object of class nb_cell
- Optional input (...'propertyName',PropertyValue,...):
 - varargin :
 - > 'saveName' : The data is saved to a excel worksheet with the same name as the input after 'saveName'. If consisting of more dataset, each page of the data property is saved with the same name as the input after 'saveName' pluss the page number.
 - > 'ext' :
 - > 'xlsx' : Default. Save data to excel spreadsheet(s)
 - > 'matold' : Save the data down to a mat file. The data is saved down as a structure, with the variables

as the fieldnames, and the data as its fields.
If the data consist of more pages, each field
will consist of the same number of pages.

The types of the object is also added to the
field 'types' of the saved structure. Which
makes it possible to load the .mat field up
again to nb_cs object.

See the toStructure method with input old set to 1.

Must be combined with the optional input
'saveName'.

> 'mat' : Save the data down to a mat file. The data is
saved down as a structure, with the variables stored
in the field 'variables', while the data of the
variables will be saved as its fields with generic
names ('Var1','Var2' etc). If the data consist
of more pages, each field will consist of the same
number of pages.

The types of the object is also added to the
field 'types' of the saved structure. Which
makes it possible to load the .mat field up
again to nb_cs object.

See the toStructure method with input old set to 1.

Must be combined with the optional input
'saveName'.

> 'mats' : The object is saved down as a struct using
the nb_cell.struct function.

> 'txt' : Save the data to .txt file.

> 'append' :

> 1 : Saves all the datasets (pages of the data) to one
excel spreadsheet, but in seperate worksheets.
(With the names of the dataset, given by dataNames
property of the object.)

Works only when 'ext' input is set as 'xlsx'. (The
default)

Caution : If saved in this way, it is not possible
to load it up again to a nb_cs object

> 0 : The default saving method.

> 'sheets' : A cellstr with the sheet names to save the datasets/pages
of the object down to. Must match the dataNames property.
This option has precedence over 'append'.

Output:

Saved output in the wanted format.

See also:

[asCell](#), [toStructure](#), [nb_readMat](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

► **sec ↑**

`obj = sec(obj)`

Description:

Secant of argument in radians.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

`obj = sec(obj);`

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **secd ↑**

`obj = secd(obj)`

Description:

Secant of argument in degrees.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **setInfinite2NaN** ↑

```
obj = setInfinite2NaN(obj)
```

Description:

Set all elements that are isfinite to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setMethodCalls** ↑

```
obj = setMethodCalls(obj,c)
```

Description:

Set all method calls of the object with a cell matrix. The object needs to be updatable.

Caution: To delete method calls use deleteMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.
- c : A cell matrix. See the output of getMethodCalls

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

See also:

[nb_cell.deleteMethodCalls](#), [nb_cell.getMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setValue** ↑

```
obj = setValue(obj,columns,Value,rows,pages)
```

Description:

Set some values of the dataset(s). (Only one column.)

Input:

- obj : An object of class nb_cs
- column : The index of the column you want to set some values of. As an integer.
- Value : The values you want to assign to the column. Must be a cell vector (with the same number of pages as given by pages or as the number of dataset the object consists of.) Must have the same length as the input

given by rows. (If this is not provided this input must the same number of rows as the cdata property)

- rows : A vector of row indexes to set.
- pages : At which pages you want to set the values of a column. Must be a numerical index of the pages you want to set.
E.g. if you want to set the values of the 3 first datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages must be set.

Output:

- obj : An nb_cell object with the added variable

Written by Kenneth S. Paulsen

► **setZero2NaN** ↑

obj = setZero2NaN(obj)

Description:

Set all elements that are 0 to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sin** ↑

obj = sin(obj)

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **sind** ↑

```
obj = sind(obj)
```

Description:

sind(obj) is the sine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

sinh(obj) is the hyperbolic sine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = sinh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sortProperty** ↑

obj = sortProperty(obj,type)

Description:

Sort the variables/types/datasets of the object alphabetically.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sqrt** ↑

obj = sqrt(obj)

Description:

sqrt(obj) is the square root of the elements of obj. Complex results are produced for non-positive elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **stopSorting** ↑

```
obj = stopSorting(obj)
```

Description:

Stop sorting of variables.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where sorting is turned off.

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **struct** ↑

```
s = struct(obj)
```

Description:

Object to struct.

Input:

- obj : An nb_cell object.

Output:

- s : A structure representing an nb_cell object.

Written by Kenneth SÃ¶therhagen Paulsen

► **subsasgn** ↑

```
varargout = subsasgn(obj,s,b)
```

Description:

Makes it possible to do subscripted assignment of the data of a nb_cell object

Input:

- obj : An object of class nb_cell

- s : The same input as when you do subscripted assignment one a cell matrix

- b : The assign data at the given index s.

Output:

- obj : An nb_cell object where the assign data property is set.

Examples:

```
obj(1:2) = {2;3};
```

same as

```
obj(1:2,1,1) = {2;3};
```

same as

```
obj.data(1:2,1,1) = {2;3};
```

```
obj(1:2,2,3) = {2;'Test'};
```

Written by Kenneth S. Paulsen

► **subsref** ↑

Description:

Makes it possible to do subscripted reference of the data of a nb_cell object. See the examples for more

Uses the method window, just using indexing instead

Input:

- obj : An object of class nb_cell
- s : A structure on have to index the object

Output:

- obj : An nb_cs object indexed as wanted.

Examples:

obj = obj(1,2:4,1)

same as

obj = obj(1,2:4)

Written by Kenneth S. Paulsen

► **tail ↑**

tail(obj,nRows,page)

Description:

Display the last n rows of a nb_cell object.

Input:

- obj : An nb_cell object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_cell](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **tan ↑**

```
obj = tan(obj)
```

Description:

Take the tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = tan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **tand** ↑

```
obj = tand(obj)
```

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

tanh(obj) is the hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **toMfile** ↑

```
string = toMfile(obj)
toMfile(obj,filename)
string = toMfile(obj,filename,varName)
```

Description:

Write a .m file to generate the current object.

Input:

- obj : An nb_cell object.

- filename : The filename to write the code to. If empty no file will be written.

- varName : Name of decleared variable by this code. Default is data.

Output:

- string : A cellstr with the .m code to generate the current object.

Written by Kenneth Sæterhagen Paulsen

► **toStructure** ↑

```
retStruct = toStructure(obj)
```

Description:

Transform the obj to a structure.

Input:

- obj : An object of class nb_cell
- old : Indicate if old mat file format is wanted. Default is 0 (false).

Output:

- retStruct : A structure.

See also:

[nb_cs.struct](#)

Written by Kenneth S. Paulsen

► **transpose ↑**

```
obj = transpose(obj)
```

Description:

Take the transpose (.)' of an nb_cell object.

Input:

- obj : An object of class nb_cell

Output:

- obj : An nb_cell object which represents the transposed input object

Written by Kenneth S. Paulsen

► **uminus ↑**

```
obj = uminus(obj)
```

Description:

Unary minus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : A nb_ts, nb_cs or nb_data object where all the data are the unary minus of the input objects data

Examples:

```
obj = -obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **unstruct** ↑

```
obj = nb_cell.unstruct(s)
```

Description:

Reverse of the struct method.

Input:

- s : See the s output of the struct method.

Output:

- obj : An nb_cell object.

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,warningOff,inGUI)
```

Description:

This method will try to update the data of the nb_ts object.

Caution : It is only possible to update an nb_dataSource object which has updateable links to a FAME database or a full directory (path) to an excel spreadsheet or .mat file or

the SMART database.

Caution : If you want to create a link to a specific worksheet of a excel file you must provide the extension!

Caution : All method calls done on the object are preserved.
(There exist some exception; and, or etc.)

Input:

- obj : An object of class nb_dataSource which are updatable.
- warningOff : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.

Output:

- obj : An object of class nb_dataSource with the data updated. If it is not possible a warning will be given.

Examples:

```
obj = nb_ts(['N:\Matlab\Utvikling\MATLAB_LIB\NEMO\'...  
            'Documentation\Examples\example_ts_quarterly']);  
obj = obj.update
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **uplus ↑**

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE
```

► **vertcat** ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_cell
- b : An object of class nb_cell
- varargin : Optional numbers of objects of class nb_cell

Output:

- a : An nb_cell object where the data from the different objects are appended to each other.

Written by Kenneth S. Paulsen

► **window** ↑

```
obj = window(obj,rows,columns,pages)
```

Description:

Narrow down the dataset(s) of the nb_cell object

Input:

- obj : An object of class nb_cell
- rows : The row indexes. As a 1 x N integer.
- columns : The columns indexes. As a 1 x N integer.
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty all pages is kept.

```
I.e. obj = obj.window('','','',1:3)
```

Or it can be the name of the dataset you want to keep. Then the input must be a string.

```
I.e. obj = obj.window('','','','Database1')
```

Or it can be a cell array of the datasets names to keep.

```
I.e obj = obj.window('','','',{ 'Database1',...  
'DataBase2'})
```

Output:

- obj : An nb_cell object where the datasets of the input object are narrowed down to the given window.

Written by Kenneth S. Paulsen

► **writeTex ↑**

```
writeTex(obj,filename)  
writeTex(obj,filename,'inputName', inputValue,...)
```

Description:

Writes the object data to a LaTeX table saved to a .tex file.

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_cell
- filename : The name of the saved .tex file. With or without the extension.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:

```
'Huge', 'huge', 'LARGE', 'Large', 'large',
'normalsize' (default), 'small',
'footnotesize', 'scriptsize','tiny'.
```

Caution : This option is case sensitive.

- 'justification' : Sets the justification of the caption of the table. Must be a string.

```
Either 'justified','centering' (default),
'raggedright', 'RaggedRight', 'raggedleft'.
```

Caution : This option is case sensitive.

- language : Either 'english' (default) or 'norwegian' ('norsk').

- lookUpMatrix :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
lookUpMatrix = {

'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined char, which will result in multi-line text of the table.

Caution : This will act on both variables and types.

- 'precision' : Sets the precision of the numbers in the table. Must be a string. Default is '%3.2f'. I.e. two decimals.

- 'rotation' : The rotation of the table in LaTeX. Either:

```
> 'sidewaystable' : A sideways table
> 'landscape'      : Rotate the whole page
> ''               : No rotation
```

- 'rowColor1' : Color of every second row of the table starting from the second row. Same options as was the case for the 'firstRowColor' option.

- 'rowColor2' : Color of every second row of the table starting from the third row. Same options as was the case for the 'firstRowColor' option.

Output:

A saved .tex file with the LaTeX table

Examples:

```
obj = nb_cell({'Test',2; 'Test2', 3});  
obj.writeTex('test')
```

See also:

[nb_cell](#)

Written by Kenneth Sæterhagen Paulsen

► **zeros** ↑

```
obj = nb_cell.zeros(rows,columns)  
obj = nb_cell.zeros(rows,columns,pages)
```

Description:

Create an nb_cell object with all elements set to 0.

Input:

- rows : Number of rows of the nb_cell object.
- vars : Number of columns of the nb_cell object.
- pages : Number of pages of the nb_cell object.

Output:

- obj : An nb_cell object.

See also:

[nb_cell](#)

Written by Kenneth Sæterhagen Paulsen

■ **nb_cs**

Go to: [Properties](#) | [Methods](#)

A class for storing cross-sectional data (Case data)

Constructor:

```
obj = nb_cs(datasets,NameOfDatasets,types,variables)
obj = nb_cs(datasets,NameOfDatasets,types,variables,sorted)
```

Input:

- datasets:

Input can be one of the data types listed below:

- xls(x) : A name of the excel spreadsheet to import. (With or without extension.). As a string.

Caution : If you read a specific worksheet, and want the object to be updateable you must provide the extension.

- mat : Name(s) of the .mat file(s) to import. (With or without extension.). As a string. This .mat file must store a struct where each field stores the data of a variable. The data stored at each field must be doubles with same size. The only exception is that you must include a field called 'types', storing a cellstr with the types of the data. (This cellstr must have the same size as the provided data (size of the first dimension)).

Caution : The data stored as seperate field could have more pages (size of the third dimension could be lager then 1). Again All the data of the variables must be of the same size.

- double : As a double matrix. Each page of the matrix will be added as separate datasets.

- struct : As a structure of double matrices. Each field is added as a separate datasets.

You can also give a cell array consisting of one or a combination of the data types mentioned above. Each cell will be added as a new dataset.

Extra: Use the method structure2nb_cs function to add all the fields of the structure as variables in an nb_cs object. (If each field has more page then 1, then each page will be a new datasets). See the method structure2nb_cs for more on the supported format of the structure you can provide as input.

- NameOfDatasets :

Input will depend on the data type provided to the datasets input:

- xls(x) : A string with the wanted dataset name. Default is the excel file name. If the provided string is a sheet of the read spreadsheet that specific worksheet will be read.

- mat : A string with the wanted dataset name. Default is the .mat file name.

- double : A string with the wanted dataset name. Default is 'Dataset1'.

Caution: When you given double matrices which has more pages then one, either directly or through a struct. Each of the pages will be added as a dataset. The name of this datasets will, if the input NameOfDatasets has the same size as the input datasets, get the name NameOfDatasets{jj}<jj>, where jj is the page number, or else get the name 'Database<jj>'.

- struct : This input has nothing to say. The dataset names will be given by the fieldnames.

- A cell array of the listed types above:

If you give a cell array to the datasets input, this input must be a cell array of strings. If you give a cell array which doesn't have the same size as the cell array given to the datasets input, the datasets which is left gets the name 'Database<jj>', where <jj> stands for this added datasets number.

If you give an empty cell array, i.e. {} all the datasets get the names; 'Database<jj>', where <jj> stands for the added datasets number. Except when the datasets input are given by a cellstr with the excel file names or .mat file names, then the default dataset names will be set to the same cellstr. Or if some elements of the cell is a struct, then the datasets provided through the struct will get their fieldnames as the dataset names.

- types :

A cell array of the type names of the data, as strings. Only needed when you give a double or a struct of doubles as the datasets input. The number of types must be the same as the size of the data (1- dimension).

- variables :

A cell array of the variable names of the data, as strings. Only needed when you give a double or a struct of doubles as the datasets input. The number of variables must be the same as the size of the data (2- dimension).

- sorted : true or false. Default is true.

Output:

- obj : An object of class nb_cs.

Examples:

```

- xls(x):
    One excel spreadsheet
    obj = nb_cs('test1')

    More excel spreadsheets
    obj = nb_cs({'test1','test2',...})
    same as
    obj = nb_cs({'test1','test2',...},{'test1','test2',...})

- mat:
    One .mat file:
    obj = nb_cs('test1')

    More .mat files:
    obj = nb_cs({'test1','test2',...})
    same as
    obj = nb_cs({'test1','test2',...},{'test1','test2',...})

-struct:
    One struct (Consisting of double matrices):
    obj = nb_cs(struct1,{},{'Type1','Type2',...},...
                {'Var1','Var2',...})

    More structs:
    obj = nb_cs({struct1,struct2,...},{},{'Type1','Type2',...},...
                {'Var1','Var2',...})

- double matrix:
    One matrix
    obj = nb_cs(double1,'',{'Type1','Type2',...},...
                {'Var1','Var2',...})

    obj = nb_cs(double1,'test1',{'Type1','Type2',...},...
                {'Var1','Var2',...})

    More matrices
    obj = nb_cs({double1,double2,...},{},...,
                {'Type1','Type2',...},{'Var1','Var2',...})

    obj = nb_cs({double1,double2,...},{'name1','name2',...},...
                {'Type1','Type2',...},{'Var1','Var2',...})

```

One matrix with more pages:

```
obj = nb_cs(double1, {}, {'Type1','Type2','...'},...
             {'Var1','Var2','...'})

obj = nb_cs(double1, {'name1','name2','...'},...
             {'Type1','Type2','...'}, {'Var1','Var2','...'})
```

See also:

[nb_graph_cs](#), [nb_ts](#), [nb_graph_ts](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- **data**
- **dataNames**
- **localVariables**
- **numberOfDatasets**
- **numberOfTypes**
- **numberOfVariables**
- **types**
- **userData**
- **variables**

- **data** [↑](#)

The data of the object. As a double or logical.

Inherited from superclass NB_DATASOURCE

- **dataNames** [↑](#)

The names of the different dataset. As a cellstr.

Inherited from superclass NB_DATASOURCE

- **localVariables** [↑](#)

A nb_struct with the local variables of the nb_ts object.
I.e. a field with name 'test' can be reach with the string
input %#test. Only for dates and vintage inputs.

Inherited from superclass NB_DATASOURCE

- **numberOfDatasets** [↑](#)

Number of datasets stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **numberOfTypes** [↑](#)

Number of types of the object. As a double.

- **numberOfVariables** [↑](#)

Number of variables stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **types** [↑](#)

Description/names of the data types. As a cellstr.

- **userData** ↑

Add user data. Can be of any type.

Inherited from superclass NB_DATASOURCE

- **variables** ↑

Variables of the object. As a cellstr.

Inherited from superclass NB_DATASOURCE

Methods:

- abs
- acosh
- acoth
- acsch
- addPageCopies
- addPrefix
- and
- asCellForMPRReport
- asech
- asinh
- atand
- callfun
- ceil
- cos
- cot
- cov
- csc
- ctranspose
- cumsum
- deleteVariables
- diag
- empty
- exp
- extMethod
- get
- getCreatedVariables
- getSourceList
- acos
- acot
- acsc
- addDataset
- addPages
- addType
- appendFromAnotherPage
- asec
- asin
- assignNaN
- atanh
- callop
- conj
- cosd
- cotd
- createType
- cscd
- cumnansum
- deleteMethodCalls
- demean
- disp
- eq
- expand
- floor
- getClassOfData
- getMethodCalls
- getVariable
- acosd
- acotd
- acscd
- addDatasets
- addPostfix
- addVariable
- asCell
- asecd
- asind
- atan
- breakLink
- callstat
- corr
- cosh
- coth
- createVariable
- csch
- cumprod
- deleteTypes
- deptcat
- double
- estimateDist
- expm1
- ge
- getCode
- getSource
- gt

- hasVariable
- horzcat
- interwine
- isDistribution
- isUpdateable
- isnan
- keepTypes
- le
- log1p
- map2Distribution
- median
- min
- mode
- mtimes
- not
- or
- permute
- pow2
- rdivide
- realpow
- renameMore
- saveDataBase
- sech
- setValue
- sind
- sort
- sqrt
- head
- interpolate
- isBoolean
- isDouble
- isempty
- isscalar
- keepVariables
- log
- log2
- max
- merge
- minus
- mpower
- nan
- objSize
- packing
- plot
- power
- real
- realsqrt
- reorder
- sec
- setInfinite2NaN
- setZero2NaN
- sinh
- sortProperty
- squeeze
- histogram
- intertwine
- isCrossSectional
- isTimeseries
- isfinite
- keepPages
- kurtosis
- log10
- lt
- mean
- merge2Series
- mldivide
- mrdivide
- ne
- ones
- percentiles
- plus
- rand
- reallog
- rename
- round
- secd
- setMethodCalls
- sin
- skewness
- sortTypes
- std

- stopSorting
 - struct
 - structure2Dataset
 - subsasgn
 - subsref
 - sum
 - summary
 - tail
 - tan
 - tand
 - tanh
 - times
 - toMFile
 - toStructure
 - tonb_cell
 - tonb_data
 - tonb_ts
 - transpose
 - uminus
 - unstruct
 - update
 - uplus
 - var
 - vertcat
 - window
 - writeTex
 - zeros
-

► **abs** ↑

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acos** ↑

```
obj = acos(obj)
```

Description:

Take acos of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acos(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acosd** ↑

obj = acosd(obj)

Description:

acosd(obj) is the inverse cosine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acosh** ↑

obj = acosh(obj)

Description:

acosh(obj) is the inverse hyperbolic cosine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acosh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acot** ↑

```
obj = acot(obj)
```

Description:

Take acotangent of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = acot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

acotd(obj) is the inverse cotangent, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acotd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acoth** ↑

obj = acoth(obj)

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acoth(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsc** ↑

obj = acsc(obj)

Description:

Take inverse csc of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acsc(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acscd** ↑

obj = acscd(obj)

Description:

acscd(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acscd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsch** ↑

obj = acsch(obj)

Description:

acsch(obj) is the inverse hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **addDataset ↑**

```
obj = addDataset(obj,dataset,NameOfDataset,types,variables)
```

Description:

Makes it possible to add one dataset to a existing nb_cs object

Caution : If more than one dataset is added to an object the link to the data source is broken!

Input:

- obj : An object of class nb_cs

- dataset : Either a string with the name of excel spreadsheet or a .mat file (no extension needed), or a numerical matrix.

- NameOfDataset : Must be a string with the dataset name. If not given (or given as '') the default name Database<jj> is given. Where jj is the number of added datasets of the object, including dataset you add with this method.

- types : A cell array of string with the names of the types you want to add. Only needed when numerical matrix of data is added to the object. Must have the same size as the data you add.

- variables : A cell array of string with the names of the variables you want to add. Only needed when numerical matrix of data is added to the object. Must have the same size as the data you add.

Output:

- obj : The object itself with the added dataset.

Examples:

```
obj = nb_cs;
obj = obj.addDataset([2,2],'test',{'First'},{'Var1','Var2'});
```

Written by Kenneth S. Paulsen

► **addDatasets** ↑

```
obj = addDatasets(datasets,NameOfDatasets,types,variables)
```

Makes it possible to add more datasets to a existing nb_cs object

Input:

Same input as of the constructor but only cell arrays are supported for the inputs 'datasets' and 'NameOfDatasets'. 'NameOfDatasets' could be given as a empty cell; {};

Output:

- obj : An object of class nb_cs, now including the new datasets provided

Examples:

```
obj = addDatasets({'excel1','excel2'},{},{'First'},...
{'Var1','Var2'})
```

Written by Kenneth S. Paulsen

► **addPageCopies** ↑

Description:

Add copies of the data of an object of class nb_cs and append it as new datasets of the object. Only possible if an object has only one page.

Input:

- obj : An object of class nb_cs
- num : Number of copies to be appended

Output:

- obj : An object of class nb_cs with the added copies of the first dataset in the object.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = addPageCopies(obj,2);
```

Written by Kenneth S. Paulsen

► **addPages** ↑

```
obj = addPages(obj,DB)
```

Description:

Add all pages from a different nb_cs object with the current one

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_cs
- DB : An object of class nb_cs
- varargin : Optional number of objects of class nb_cs

Output:

- obj : An object of class nb_cs with all the datasets from the objects obj and varargin.

Be aware: If the added datasets does not contain the same variables or types, the data of the nb_cs object with the tightest window will be expanded automatically to include all the same types and variables. (the added data will be set as nan)

Examples:

```
obj = addPages(obj,DB);
```

Written by Kenneth S. Paulsen

► **addPostfix** ↑

```
obj = addPostfix(obj,postfix)
obj = addPostfix(obj,postfix,vars)
```

Description:

Add a postfix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- postfix : The postfix to add to all the variables of the object. Must be a string
- names : A cellstr of the variables/types to add the prefix to. Default is all variable of the object.
- type : Either:
 - 'variables' : To rename variables (default)
 - 'types' : To rename types

Output:

- obj : An object of class nb_ts, nb_cs, nb_bd or nb_data with the postfix added to the variables selected.

Examples:

```
obj = obj.addPostfix('_NEW');
```

Written by Kenneth S. Paulsen

► **addPrefix ↑**

```
obj = addPrefix(obj,prefix)
obj = addPrefix(obj,prefix,vars)
```

Description:

Add a prefix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- prefix : The prefix to add to all the variables of the object. Must be a string
- vars : A cellstr of the variable to add the prefix to. Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with a prefix added to the selected variables

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► addType ↑

```
obj = addType(obj,type,addedData,vars)
```

Description:

Add a new type to the datasets (including all pages)

Input:

- obj : An object of class nb_cs
- type : The type to add. As a string. Cannot be the same as a existing type. Use setValue instead.
- addedData : The data of the added type. Must match the number of variables given. I.e. a double with size 1 x number of variables given in the cellstr variables.
- vars : A cellstr of the variables of the added type.

Output:

- obj : An nb_cs object with the added type

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.addVariable('Second',[2,2],{'Var1','Var2'})
```

Written by Kenneth S. Paulsen

► addVariable ↑

```
obj = addVariable(obj,Types,Data,variable)
```

Description:

Add a new variable to the datasets (including all pages)

Input:

- obj : An object of class nb_cs
- Types : The types of the data of the added variable.
Must be a cellstr.
- Data : The data of the added variable. Must match the number of types given.
- Variable : A string with the added variable name. Cannot be the same as a existing variable. (Use setValue instead)

Output:

- obj : An nb_cs object with the added variable

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.addVariable({'First'},2,'Var3')
```

Written by Kenneth S. Paulsen

► and ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the and operator has evaluated all the data elements of the object.
The data will be a logical matrix

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **appendFromAnotherPage** ↑

```
obj = appendFromAnotherPage(obj,page)
```

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_dataSource object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_dataSource object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **asCell** ↑

```
cellMatrix = asCell(obj,page)
```

Description:

Return the nb_cs objects data as a cell matrix.

Input:

- obj : An object of class nb_cs
- page : Which pages should be transformed to a cell. Must be a double (vector) with the indicies.

Output:

- cellMatrix : The objects data transformed to a cell. (With types and variable names)

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
cellMatrix = obj.asCell();
```

Written by Kenneth S. Paulsen

► **asCellForMPRReport** ↑

```
cellMatrix = asCellForMPRReport(obj,roundoff)
```

Description:

Return the nb_cs objects data as a cell matrix, on the MPR publishing format

Input:

- obj : An object of class nb_cs
- roundoff : 1 if you want to round off to 2 decimals, 0 otherwise. 1 is default.

Output:

- cellMatrix : The objects data transformed to a cell. (With types and variable names) Only the first page is returned. Look very much like an excel spreadsheet.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
cellMatrix = asCellForMPRReport(obj,1);
```

Written by Kenneth S. Paulsen

► **asec** ↑

```
obj = asec(obj)
```

Description:

Take inverse sec of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asecd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sin of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► asinh ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► assignNan ↑

```
obj = expand(obj,variables,value)
```

Description:

Assign all the nan observation to the value input.

Input:

- obj : An object of class nb_dataSource.
- variables : A char or a cellstr with the variables to assign nan values.
- value : A scalar number.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj      = nb_ts.rand('2012Q1',10,2,3);
obj(1:2,1,:) = nan;
obj      = assignNan(obj,'Var1',0)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the inverse tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees, of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = atand(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **atanh** ↑

obj = atanh(obj)

Description:

atanh(obj) is the inverse hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = atanh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **breakLink** ↑

obj = breakLink(obj)

Description:

Break link to source

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callfun ↑**

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s). The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb.DataSource.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nb.DataSource.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb.DataSource objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource object. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```

d      = nb_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)

```

See also:

[nb_cs.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callop** ↑

```

obj = callop(obj,another,func)
obj = callop(obj,another,func,type)

```

Description:

Call mathematical operator on the data of the object. In contrast to calling the operator itself on the object, this method does not try to match the same variables from the two inputs, but instead operate in the same way as mathematical operators do on double matrices (bsxfun). The time domain is matched though!

Caution: The following operators are supported:

- > @plus or 'plus'
- > @minus or 'minus'
- > @times or 'times'
- > @rdivide or 'rdivide'
- > @power or 'power'

Input:

- obj : An object of class nb_dataSource.
- another : An object of class nb_dataSource.
- func : One of the above listed functions.
- type : Either:
 - > 'keep' : Keep the variable names of the object with most variables. If they have the same number of variables, they are taken from the first object.
 - > 'rename' : The new variable names represent the operation that has been done to the separate series. So if you divide an object with one series named 'Var1' with another object with one series named 'Var2', the new name will be 'Var1./Var2'. (Default)

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource method. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,1),'''2011Q1',{'Var2'});
d3 = callop(d1,d2,@minus);

d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,2),'''2011Q1',{'Var2','Var3'});
d3 = callop(d1,d2,@minus);
```

See also:

[nb_cs.getData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► callstat ↑

```
obj = callstat(obj,func)
obj = callstat(obj,func,varargin)
```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s) that goes from the data of the object have more than one variable to only having one. E.g. when taking the mean over a set of variables.

Input:

- obj : An object of class nb.DataSource.

- func : A function handle (or one line char) that maps a nObs x nVar x nPage double to nObs x 1 x nPage double, or that maps a nObs x nVar x nPage double to nObs x nVar x 1 double.

Optional input:

- 'name' : Set the name of the new variable, as a one line char. Default is to use the str2func on the provided function handle.

- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- obj : An object of class nb_dataSource with only one variable.

Examples:

```
d = nb_ts.rand('2012Q1',10,3);
d1 = callstat(d,@(x)mean(x,2),'name','mean')
```

See also:

[nb_cs.mean](#), [nb_cs.std](#), [nb_cs.var](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► ceil ↑

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = ceil(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► conj ↑

```
obj = conj(obj)
```

Description:

conj(obj) is the complex conjugate of the elements of obj.
For a complex element x of obj, conj(x) = REAL(x) - i*IMAG(x).

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = conj(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **corr ↑**

```
obj = corr(obj,varargin)
```

Description:

Calculate the correlation of the series of the nb_cs object

Caution: Calculates the correlation matrix of the variables of the nb_cs object, if no optional inputs are given.

Input:

- obj : An object of class nb_cs

Output:

- obj : An nb_cs object with the wanted correlation

Examples:

```
corrMatrix = corr(obj);
```

Written by Kenneth S. Paulsen

► **cos ↑**

```
obj = cos(obj)
```

Description:

Take cos of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = cos(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cosd** ↑

obj = cosd(obj)

Description:

cosd(obj) is the cosine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = cosd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cosh** ↑

obj = csch(obj)

Description:

cosh(obj) is the hyperbolic cosine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = csch(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cot** ↑

obj = cot(obj)

Description:

Take cotangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = cot(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cotd** ↑

obj = cotd(obj)

Description:

cotd(obj) is the cotangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **coth** ↑

```
obj = coth(obj)
```

Description:

coth(obj) is the hyperbolic cotangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = coth(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cov** ↑

```
obj = cov(obj,varargin)
```

Description:

Calculate the covariances of the timeseries of the nb_cs object

Caution: Calculates the covariance matrix of the variables of the nb_cs object, if no optional inputs are given. If 'lags' is given this function calculates the covariance with the number of wanted lags for each variable with itself. (And only with itself)

Input:

- obj : An object of class nb_cs
- Optional input ('propertyName',PropertyValue):
 - 'lags' : An integer with the number of lags you want to calculate the covariance for. (Here for each variable)

Output:

- obj : An nb_cs object with the wanted covariances

Examples:

```
covMatrix = cov(obj);
```

Written by Kenneth S. Paulsen

► createType ↑

```
obj = createType(obj, nameOfNewType, expression)
```

Description:

Create type(s) and add them to the nb_cs object dataset(s)

Caution :

- This function does not allow the type names to include mathematical operators (defined in matlab) as for example - + / . () !
- This function only support methods of the class double for the expressions. I.e. all the types of the nb_ts object are dumped as doubles, which you can use to create a new type

Input:

- obj : An object of class nb_cs
- nameOfNewType : The created types names as a cell array or a string.
 - Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the data of all the created variables.
 - As a cell array or a string.
 - Must have the same size as the 'nameOfNewType' input.

Output:

- obj : An nb_cs object with the created types(s) added.

Examples:

```
obj = obj.createType('type1','type1./type2');
```

```
obj = obj.createType({'var3','var4',...},...  
{'type1./type2','type2./type1',...});
```

Written by Kenneth S. Paulsen

► **createVariable ↑**

```
obj = createVariable(obj,nameOfNewVariable,expression)  
obj = createVariable(obj,nameOfNewVariable,expression,parameters,varargin)
```

Description:

Create variable(s) and add them to the nb_cs object dataset(s)

Caution :

- This function only support methods of the class double for the expressions.

Input:

- obj : An object of class nb_cs
- nameOfNewVariable : The created variable names as a cell array or a string.
Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the data of all the created variables.
As a cell array or a string.
Must have the same size as the 'nameOfNewVariable' input.
- parameters : A struct with the parameters that can be used in the expressions. Caution: They will be added as series, so you need to use elementwise operators (.*, ./, .^)!

Optional input:

- 'overwrite' : true or false. Set to true to allow for overwriting of variables already in the data of the object. Default is false.
- 'warning' : true or false. Set to true to give warning if expressions fail (instead of error). Will result in series having all nan value, when warning is thrown.

Output:

- obj : An nb_cs object with the created variable(s) added.

Examples:

```
obj = obj.createVariable('var3','var1./var2');  
  
obj = obj.createVariable({'var3','var4',...},...  
{'var1./var2','var2./var1',...});
```

Written by Kenneth S. Paulsen

► **csc** ↑

```
obj = csc(obj)
```

Description:

Take csc of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cscd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csch** ↑

```
obj = csch(obj)
```

Description:

csch(obj) is the hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **ctranspose** ↑

```
obj = ctranspose(obj)
```

Description:

Take the transpose ('') of an nb_cs object

Caution : The former types will be sorted when transposing the object.

Input:

- obj : An object of class nb_cs

Output:

- obj : An nb_cs object which represents the transposed input object

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'})
```

```
obj =
```

```
'Types'      'Var1'      'Var2'  
'First'      [    2]      [    2]
```

```
t = obj'
```

```
t =
```

```
'Types'      'First'  
'Var1'       [    2]  
'Var2'       [    2]
```

Written by Kenneth S. Paulsen

► **cumnansum ↑**

```
obj = cumnansum(obj)
```

Description:

Cumulative sum of the series of the object. Ignoring nan values.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumnansum(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumprod** ↑

```
obj = cumprod(obj,dim)
```

Description:

Cumulativ product of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : A nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative product of the old objects 'data' property.

Examples:

```
obj = cumprod(obj);
obj = cumprod(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumsum(obj);  
obj = cumsum(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteMethodCalls** ↑

```
obj = deleteMethodCalls(obj,c)
```

Description:

Delete some method calls of the object. The object needs to be updatable.

Caution: To set method calls use setMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- sourceNr : The source index to delete the methods from
- methodNrs : A index with the methods to delete.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

See also:

[setMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteTypes** ↑

```
obj = deleteTypes(obj,deletedType)
```

Description:

Delete types from the current nb_cs object (letter size has something to say)

If some of the types you specify in deletedType does not exist in the obj, this will not have anything to say for the obj. (The types cannot be deleted, because the types do not exist.)

Input:

- obj : An object of class nb_cs
- deletedType : Must be a char or a cellstr

Delete all the type names, given in the char array or cellstr array deletedType, of current the nb_cs object.

NB! If deletedType is just one string and include a * as the last letter, this function will delete all the type names, from the object, that start with the letters which is in front of the * in the string.

i.e. obj = deleteTypes(obj,'DPQ*');

Deletes all the types in the obj which has type names starting with 'DPQ'.

If deletedType is just one string and include a * as the first letter, this function will delete all the type names, from the object, that include the letters which follows in the string.

i.e. obj = deleteTypes(obj, '*shift');

Deletes all the types in the obj which has type names which include 'shift'.

Output:

- obj : A nb_cs object with the types specified in deletedType deleted.

Examples:

```
obj = nb_cs([2;2],'test',{'First','Second'},{'Var1'});  
obj = deleteTypes(obj, {'First','Second'});  
obj = deleteTypes(obj, char('First','Second'));  
obj = deleteTypes(obj, obj.types);
```

Written by Kenneth S. Paulsen

► **deleteVariables ↑**

```
obj = deleteVariables(obj,deletedVar)
```

Description:

Delete variables from the current nb_ts, nb_data or nb_cs object (letter size has something to say)

If some of the variables you specify in deletedVar does not exist in the obj, this will not have anything to say for the obj. (The variables cannot be deleted, because the variables do not exist.)

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- deletedVar : Must be a char or a cellstr

Delete all the variable names, given in the char array or cellstr array deletedVar, of current the object.

NB! If deletedVar is just one string and include a * as the last letter, this function will delete all the variable names, from the object, that start with the letters which is in front of the * in the string.

i.e. obj = deleteVariables(obj,'DPQ*');

Deletes all the variables in the obj which has variable names starting with 'DPQ'.

If deletedVar is just one string and include a * as the first letter, this function will delete all the variable names, from the object, that include the letters which follows in the string.

i.e. obj = deleteVariables(obj, '*shift');

Deletes all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_ts, nb_data or nb_cs object with the variables specified in deletedVar deleted.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = deleteVariables(obj, {'Var1','Var2'});  
obj = deleteVariables(obj, char('Var1','Var2'));  
obj = deleteVariables(obj, obj.variables);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_dataSource object.
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_dataSource object.

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATASOURCE

► **deptcat** ↑

```
obj = deptcat(obj,varargin)
```

Description:

Depth concatenation (add pages of different objects) (No short notation)

Be aware: If the added datasets does not contain the same variables or types, the data of the nb_cs object with the tightest window will be expanded automatically to include all the same types and variables. (the added data will be set as nan)

Input:

- obj : An object of class nb_cs
- varargin : Optional number of nb_cs objects

Output:

- obj : An object of class nb_cs where all the objects datasets are added into.

Examples:

```
obj = obj.deptcat(DB);  
obj = obj.deptcat(DB,DB2,DB3);
```

See also:

addPages

Written by Kenneth S. Paulsen

► **diag ↑**

```
obj = diag(obj)
```

Description:

Take the diagonal elements of the data and return it as a own type.

The data must be symmetric matrix!

Input:

- obj : An object of class nb_cs.
- typeName : Name of the created type. Default is 'diag'.

Output:

- obj : An object of class nb_cs with one type.

Written by Kenneth Sæterhagen Paulsen

► **disp ↑**

```
disp(obj)
```

Description:

Display the object on the command line (Without semicolon)

Input:

- obj : An object of class nb_cs

Output:

The nb_cs object displayed on the command line if you let out the semicolon at the end of the code line

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'})
```

Written by Kenneth S. Paulsen

► **double** ↑

```
dataOfObject = double(obj)
```

Description:

Get the data of the object as a double matrix

Caution : If the data of the object is represents as an object of class nb_distribution the mean is returned.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- dataOfObject : The data of the nb_ts, nb_cs or nb_data object

Examples:

```
dataOfObject = double(obj)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **empty** ↑

```
obj = empty(obj)
```

Description:

Empty the nb_cs object

Input:

- obj : An object of class nb_cs

Output:

- obj : An empty nb_cs object

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = empty(obj);
```

Written by Kenneth S. Paulsen

► eq ↑

```
a = eq(a,b)
```

Description:

Test if the one object is equal (==) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► estimateDist ↑

```
dist = estimateDist(obj,type)
```

Description:

Estimate distributions of the series in the nb_cs object.

Caution : This method strip all nan values.

Input:

- obj : An object of class nb_cs.
- type : The type of distribution to estimate. See the 'type' property of the nb_distribution class for the supported types. (Some may not be possible to estimate using 'mle' and/or 'mme'.) Default is 'normal';
 - Caution : If estimator is equal to 'kernel', this input is not used.
 - Caution : If type is set to 'hist', no estimator is used but the returned nb_distribution object instead represent a histogram.
- estimator : Either 'mle' (maximum likelihood), 'mme' (methods of moments) or 'kernel' (normal kernel density estimation). Default is 'mle'.
- dim : The dimension to estimate densities over. Either 1, 2 or 3. Default is 1.
- output : Either 'nb_distribution' (default) or 'nb_dataSource'. If 'nb_dataSource' is chosen the output will be a nb_cs object.

Optional input:

- varargin : Optional input given to the nb_ksdensity function. Please see help for that function.

Output:

- dist : An object of class nb_distribution or nb_cs.

See also:

[nb_distribution](#)

Written by Kenneth Sæterhagen Paulsen

► [exp ↑](#)

obj = exp(obj)

Description:

Take exp of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **expand ↑**

```
obj = expand(obj,newTypes,type,warning)
```

Description:

Expand the current nb_cs object with more types and data

Input:

- obj : An object of class nb_cs
- newTypes : The wanted new start date of the data.
- type : Type of the appended data :
 - 'nan' : Expanded data is all nan (default)
 - 'zeros' : Expanded data is all zeros
 - 'ones' : Expanded data is all ones
 - 'rand' : Expanded data is all random numbers
- warningOff : Give 'off' to suppress warning if no expansion has taken place. All 'newTypes' are already stored in the object.

Output:

- obj : An nb_cs object with expanded types.

Written by Kenneth S. Paulsen

► **expml** ↑

```
obj = expml(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, e to the `obj`.
`expml(obj)` computes `exp(obj)-1`, compensating for the roundoff in
`exp(obj)`.
(For small real `X`, `expml(X)` should be approximately `X`, whereas the
computed value of `EXP(X)-1` can be zero or have high relative error.)

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data` where the
data are equal to `e.^obj.data-1`

Examples:

```
obj = expml(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **extMethod** ↑

```
obj = extMethod(obj,method,variables,postfix,varargin)
```

Description:

This method can call a `nb_dataSource` method on a selected variables, and merge the result with the old object by adding a postfix to the new variables.

This method is the same as calling:

```
obj1 = window(obj,'','variables');
method = str2func(method);
obj = method(obj,varargin{:});
obj1 = addPostfix(obj1,method);
obj = merge(obj,obj1);
```

But without replicating the `links` property unnecessary many times!

Caution : If the object is not updateable this will result in the same as
the code above!

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- method : A str with the method to call. Must be a method of the nb_ts, nb_data or nb_cs class that does not change other dimensions than the second dimension!
- variables : A cellstr with the variables of the object to call the method on. Must be included in the object.
- postfix : A string with the postfix. If empty, the old variables will be replaced by the new ones.

Optional inputs:

These will be the inputs casted to the method except the object itself.

Extra for nb_ts and nb_data:

- '<start>' : The start date/obs of the function evaluation. Given as the second input to the window method call.
- '<end>' : The end date/obs of the function evaluation. Given as the third input to the window method call.

Output:

- obj : A nb_ts, nb_data or nb_cs object

Examples:

```
obj = nb_ts.rand(1,10,3);
obj = extMethod(obj,'lag',{'Var1'},'lag1',1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► floor ↑

```
obj = floor(obj)
```

Description:

floor(obj) rounds the data elements of obj to the nearest integers towards minus infinity

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = floor(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **ge** ↑

```
a = ge(a,b)
```

Description:

Test if the one object is greater than or equal to (\geq) the other object elementwise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.

- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get the properties of the object

Input:

- obj : An object of class nb_ts
- propertyName : Name of the property you want to get:
 - 'data' : The data of the object as a double
 - 'dataNames' : Name of the object datasets, as cellstr array
 - 'numberOfDatasets' : Number of datasets (pages) of the object. Size of the third dimension
 - 'numberOfTypes' : Number of types of the object. Size of the first dimension
 - 'numberOfVariables' : Number of variables of the object. Size of the second dimension
 - 'types' : The types of the object as a cellstr array
 - 'variables' : The variables of the object, given as a cellstr array
 - 'links' : A structure of the data source links.
 - 'sorted' : true or false
 - 'updateable' : 1 if updateable, 0 otherwise.

Output:

propertyValue : The property value of the object for the wanted property

Examples:

```
dataAsDouble = get(obj,'data');
variables     = get(obj,'variables')
```

Written by Kenneth S. Paulsen

► getClassOfData ↑

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_dataSource.

Output:

- cl : Name of the class that the data of the object is stored as.

See also:

[nb_cs.isDistribution](#), [nb_cs.isDouble](#)
[nb_cs.isBoolean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getCode ↑

```
code = getCode(obj,preamble)
code = getCode(obj,preamble,'inputName',inputValue,...)
```

Description:

Get the LaTeX code of a table representing the data of the object

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_cs
- preamble : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'combine' : If the nb_cs object consist of two or three pages (datasets) this option can be used to combine all the data in one table. Either 1 or 0. Default is 0.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:
'Huge', 'huge', 'LARGE', 'Large', 'large',
'normalsize' (default), 'small',
'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.

- 'justification' : Sets the justification of the caption of the table. Must be a string.

Either 'justified', 'centering' (default), 'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.

- 'language' : Either 'english' (default) or 'norwegian' ('norsk').

- 'lookUpMatrix' :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
lookUpMatrix = {
```

```
'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined char, which will result in multi-line text of the table.

Caution : This will act on both variables and types.

- 'orientation' : The orientation of the table. Either:

> 'horizontal' : The dates as columns
> 'vertical' : The types as columns

- 'precision' : Sets the precision of the numbers in the table. Must be a string. Default is '%3.2f'. I.e. two decimals.

- 'rotation' : The rotation of the table in LaTeX. Either:

> 'sidewaystable' : A sideways table
> 'landscape' : Rotate the whole page
> '' : No rotation

- 'rowColor1' : Color of every second row of the table starting from the second row. Same options as was the case for the 'firstRowColor' option.

- 'rowColor2' : Color of every second row of the table starting from the third row. Same options as was the case for the 'firstRowColor'

option.

- 'rowColor3' : Color of every third row of the table starting from the fourth row. Only an option in combination with the 'combine' input set to 1 and that the nb_cs object has 3 pages. Same options as was the case for the 'firstRowColor' option.
- 'NaNrepl' : Replaces NaN's in a table by chosen symbol. Give as string, can be empty. Default: 'NaN', i.e. no replacement.

Output:

- code : A char with the tex code of the table

Examples:

```
obj = nb_cs([2,2,2],'',{'type1'},{'Var1','Var2','Var3'});  
code = obj.getCode(1)
```

See also:

[nb_cs](#)

Written by Kenneth Sæterhagen Paulsen
Edited by Thor Andreas Aursland, SSB, feb. 2021: added NaNrepl option
Copyright (c) 2019, Norges Bank

► **getCreatedVariables ↑**

```
created = getCreatedVariables(obj)
```

Description:

Get a N x 2 table of the variables created using the createVariable method. The first column list the created variables, while the second column list the expressions.

Input:

- obj : An object that is a subclass of nb_dataSource.

Output:

- created : A N x 2 cell array. See description of this method for more.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getMethodCalls** ↑

```
[s,c] = getMethodCalls(obj)
```

Description:

Get all method calls as a cell matrix. The object needs to be updatable to get a list of methods called on the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.
- c : A cell matrix with a table of the method called on the object and its input. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSource** ↑

```
sourceType = getSource(obj)
sourceType = getSource(obj,type)
```

Description:

Get source type.

Input:

- obj : An object of class nb_dataSource.
- type : Give 'program' to get the general source. Default is to give the specific type of source.

Output:

- sourceType : A one line char with the source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSourceList** ↑

```
s = getSourceList(obj)
```

Description:

Get the source list of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getVariable** ↑

```
variableData = getVariable(obj, variableName, ...
                           typesOfVariable, pages)
```

Description:

Get the data of a variable of an nb_cs object

Input:

- obj : An object of class nb_cs
- variableName : The variable you want the data of. Must be given as a string (name of variable) or a cellstr.
- typesOfVariable : The wanted types, given as a cellstr.
- pages : Pages of the return data of the given variable. As a double or logical array, e.g. [1:2] or [1,3,5].

Output:

- variableData : The data of the variable you have given as a double. Return [] if not found.

Examples:

```
obj = nb_cs([2,2; 2,2],'test',{'Type1','Type2'},{'Var1','Var2'});
variableData = getVariable(obj,'Var1');
variableData = getVariable(obj,'Var1',{'Type1'});
variableData = getVariable(obj,'Var1',{'Type1'},1);
```

Written by Kenneth S. Paulsen

► **gt ↑**

```
a = gt(a,b)
```

Description:

Test if the one object is greater then (>) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a > b;
obj = 2 > b;
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hasVariable ↑**

```
[ret,location] = hasVariable(obj,variable)
```

Description:

Test if an nb_cs object has a given variable

Input:

- obj : An object of class nb_cs
- variable : The variable name as a string. Can also be a cellstr with more variables.

Output:

- ret : 1 if found otherwise 0
- location : The location of the variable if found. Otherwise [].

Examples:

```
ret           = obj.hasVariable('Var1')
[ret,location] = hasVariable(obj,'Var1')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **head** ↑

```
head(obj,nRows,page)
```

Description:

Display the first n rows of a nb_cs object.

Input:

- obj : An nb_cs object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_cs](#), [window](#), [head](#)

Written by Per Bjarne Bye

► **histogram** ↑

```
data           = histogram(obj)
[data,plotter] = histogram(obj,M)
```

Description:

Create a histogram of a object containing only one variable and one page!

Input:

- obj : A nb_dataSource object with size N x 1 x 1.
- M : The number of bins of the histogram.

Output:

- data : A nb_cs object with size M x 1 x 1.
- plotter : A nb_graph_cs object with the histogram plotted.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **horzcat** ↑

obj = horzcat(a,b,varargin)

Description:

Horizontal concatenation of nb_cs objects. ([a,b]) This is only possible if the different objects contain differen variables or have variables with the same values (types can differ). Uses the merge method.

If the types differ, nan values will be added.

Input:

- a : An object of class nb_cs
- varargin : Optional number of nb_cs objects

Output:

- obj : An nb_cs object with all the variables from the different nb_cs object merge into one dataset

Examples:

```
obj = [a,b];
obj = [a,b,c];
```

See also:

[merge](#)

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_data object. Will discard leading and trailing nan values.

Input:

- data : An nb_ts, nb_cs or nb_data object.
- method :
 - 'nearest' : Nearest neighbor interpolation
 - 'linear' : Linear interpolation. Default
 - 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - 'pchip' : Shape-preserving piecewise cubic interpolation
 - 'cubic' : Same as 'pchip'
 - 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : An nb_ts, nb_cs or nb_data object with the interpolated data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **intertwine** ↑

```
obj = intertwine(obj, cs2, '_emp', cs3, '_sim')
```

Description:

See intertwine...

Written by Henrik Halvorsen Hortemo

► **interwine** ↑

```
obj = interwine(obj, cs2, '_emp', cs3, '_sim')
```

Description:

Intertwine the current object with nb_cs objects of equal types and variables, adding postfixes to the type names of the intertwined objects.

Caution: This method will break the link to the data source.

Input:

- obj : A nb_cs object
- obj2 : A nb_cs object
- postfix2 : The postfix to add to the type names of obj2
- varargin : Supply more objects and postfixes (obj3, postfix3, ...)

Output:

- obj : An object of class nb_cs

Written by Henrik Halvorsen Hortemo

► **isBoolean** ↑

```
ret = isBoolean(obj)
```

Description:

Check if the data of the nb_dataSource object is of class logical (boolean, i.e. true or false).

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isCrossSectional** ↑

```
ret = isCrossSectional(obj)
```

Description:

Test if this object is a cross sectional data object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_cs, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isDistribution** ↑

ret = isDistribution(obj)

Description:

Check if the data of the nb_dataSource object is of class nb_distribution

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isDouble** ↑

ret = isDouble(obj)

Description:

Check if the data of the nb_dataSource object is of class double (numeric)

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isTimeseries** ↑

```
ret = isTimeseries(obj)
```

Description:

Test if a nb_dataSource object is a time series object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_ts, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isUpdateable** ↑

```
ret = isUpdateable(obj)
```

Description:

Test if this object is updateable.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : 1 if updateable, otherwise 0.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Test if each element of a nb_ts, nb_cs or nb_data object is finite.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_ts, nb_cs or nb_data object is nan.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isscalar** ↑

```
ret = isscalar(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is a scalar. Will always return 0, as an object of class nb_ts, nb_cs or nb_data is not a scalar.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : false

Examples:

```
0 = isscalar(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepPages** ↑

```
obj = keepPages(obj, pages)
```

Description:

Keep pages of the current nb_ts object

Input:

- obj : An object of class nb_ts
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty a empty nb_ts object is returned.

Can also be a cellstr with the names to keep, or a logical array with length equal to the number of pages (datasets).

Output:

- obj : An nb_ts object with the pages specified in the input are kept.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepTypes** ↑

```
obj = keepTypes(obj, keptType)
```

Description:

Keep types from the current nb_cs object (letter size has something to say)

If some of the types you specify in keptType does not exist in the object, this will not have anything to say for the object. (The types cannot be kept, because the types do not exist.)

Input:

- obj : An object of class nb_cs
- keptType : Must be a char or a cellstr

Keeps all the type names, given in the char array or cellstr array keptType, of current the nb_cs object.

NB! If keptType is just one string and include a * as the last letter, this function will keep all the type names, from the DB, that start with the letters which is in front of the * in the string.

i.e. obj = keepTypes(obj,'DPQ*');

Keep all the types in the obj which has type names starting with 'DPQ'.

If keptType is just one string and include a * as the first letter, this function will keep all the type names, from the DB, that include the letters which follows in the string.

i.e. obj = keepTypes(obj, '*shift');

Keeps all the types in the obj which has type names which include 'shift'.

Output:

- obj : A nb_cs object with the types specified in the input keptType kept, all the others are deleted.

Examples:

```
obj = nb_cs([2;2],'test',{'First','Second'},{'Var1'});  
obj = obj.keepTypes({'First','Second'});  
obj = obj.keepTypes(char('First','Second'));  
obj = obj.keepTypes(obj.types);
```

Written by Kenneth S. Paulsen

► **keepVariables ↑**

```
obj = keepVariables(obj,keptVar)
```

Description:

Keep variables from the current nb_cs object (letter size has something to say)

If some of the variables you specify in keptVar does not exist in the object, this will not have anything to say for the object. (The variables cannot be kept, because the variables do not exist.)

Input:

- obj : An object of class nb_cs
- keptVar : Must be a char or a cellstr

Keeps all the variable names, given in the char array or cellstr array keptVar, of current the nb_cs object.

NB! If keptVar is just one string and include a * as the last letter, this function will keep all the variable names, from the DB, that start with the letters which is in front of the * in the string.

i.e. obj = keepVariables(obj,'DPQ*');

Keep all the variables in the obj which has variable names starting with 'DPQ'.

If keptVar is just one string and include a * as the first letter, this function will keep all the variable names, from the DB, that include the letters which follows in the string.

i.e. obj = keepVariables(obj, '*shift');

Keeps all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_cs object with the variables specified in the input keptVar kept, all the others are deleted.

Examples:

```
obj = nb_cs([2,2,2],'test',{'First'},{'Var1','Var2','Var3'});  
obj = obj.keepVariables({'Var1','Var2'});  
obj = obj.keepVariables(char('Var1','Var2'));  
obj = obj.keepVariables(obj.variables);
```

Written by Kenneth S. Paulsen

► **kurtosis ↑**

```
obj = kurtosis(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the kurtosis of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their kurtosis.

Input:

- obj : An object of class nb_cs
 - flag :
 - > 0 : normalises by N-1 (Default)
 - > 1 : normalises by N
- Where N is the sample length.
- outputType :
 - > 'double' : Get the kurtosis values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the kurtosis values as nb_cs object.
 - dimension : The dimension to calcualate the kurtosis over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double = kurtosis(obj,0,'double');
nb_csObj = kurtosis(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► le ↑

```
a = le(a,b)
```

Description:

Test if the one object is less than or equal to (\leq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log ↑**

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on logs.

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log10 ↑**

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on (base 10) logs.

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **log2** ↑

```
obj = log2(obj)
```

Description:

`log2(obj)` is the base 2 logarithm of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **lt** ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less than (`<`) the other object elemetswise and return a `nb_dataSource` object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- `a` : An object of class `nb_dataSource` or a scalar number.

- `b` : An object of class `nb_dataSource`, but must be of same subclass as `a`, or a scalar number.

Output:

- `a` : An `nb_dataSource` object where the data element are logical.

Examples:

```
obj = a < b;
obj = 2 < b;
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► map2Distribution ↑

```
obj = map2Distribution(obj,dist)
obj = map2Distribution(obj,dist,varargin)
```

Description:

Map observation to the distributions given by dist.

The current distribution of the data is estimated using a kernel density estimator. The data must be strongly stationary. I.e. the unknown distribution from where the data has been drawn must not depend on time.

Input:

- obj : An object of class nb_dataSource.
- dist : An object of class nb_distribution with size 1 x size(obj,2).

Optional inputs:

- See the optional inputs of the nb_distribution.estimate function.

Output:

- obj : An object of class nb_dataSource.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► max ↑

```
obj = max(obj,outputType,dimension)
```

Description:

Calculate the max of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their max.

Input:

- obj : An object of class nb_cs
- outputType :
 - > 'double' : Get the max values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the max values as nb_cs object.
- dimension : The dimension to calcualate the max over.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double = max(obj,'double');
nb_csObj = max(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **mean ↑**

```
obj = mean(obj,outputType,dimension,varargin)
```

Description:

Calculate the mean of each the object. The result will be an object of class nb_cs where all the non-nan values of all the object are beeing set to their mean.

Input:

- obj : An object of class nb_cs
- outputType :
 - > 'double' : Get the mean values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mean values as nb_cs object. The output will keep the same size as the input object.
 - > 'nb_cs_scalar' : The result will be an object of class nb_cs where all the non-nan values of all the

series are being set to their mean, but where the dimension taking the mean over is reduced to size 1. The name of the type/variable/page will be 'mean'.

- dimension : The dimension to calculate the mean over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double = mean(obj,'double');
nb_csObj = mean(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **median** ↑

```
obj = median(obj,outputType,dimension,varargin)
```

Description:

Calculate the median of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their median.

Input:

- obj : An object of class nb_ts

- outputType :

> 'double' : Get the median values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.

> 'nb_cs' : Get the median values as nb_cs object.

- dimension : The dimension to calculate the median over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double    = median(obj,'double');
nb_csObj = median(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **merge ↑**

```
obj = merge(obj,DB)
```

Description:

Merge another nb_cs object with the current one

Input:

- obj : An object of class nb_cs
- DB : An object of class nb_cs

Output:

- obj : An object of class nb_cs where the datasets from the DB object are merged with the data of the obj object.

Caution:

- > It is possible to merge datasets with the same variables as long as they are representing the same data or have different types.

- > If the datasets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they can merge.

Examples:

```
obj = nb_cs([2,2; 2,2],'test',{'Type1','Type2'},{'Var1','Var2'});
DB  = nb_cs([2,1; 2,2],'test',{'Type1','Type2'},{'Var1','Var3'});
obj = obj.merge(DB)
```

Written by Kenneth S. Paulsen

► **merge2Series** ↑

```
obj = merge2Series(obj,var1,var2,new,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_dataSource.
- var1 : A one line char with the name of the base series.
- var2 : A one line char with the name of the appended series.
- new : Name of the new variable. If given as empty there will not be created a new series, but instead the var1 variable will be set to the new merged series. Default is empty.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_dataSource.

Examples:

```
obj           = nb_ts.rand('2012Q1',10,2,3);
obj(end-1:end,1,:) = nan;

obj1 = merge2Series(obj,'Var1','Var2','Var3')
obj2 = merge2Series(obj,'Var1','Var2')
obj3 = merge2Series(obj,'Var1','Var2','','diff','nLags',1)
obj4 = merge2Series(obj,'Var1','Var2','','diff','nLags',4)
obj5 = merge2Series(obj,'Var1','Var2','','level','date','2014Q1')
```

See also:

[nb_ts.merge](#), [nb_data.merge](#), [nb_cs.merge](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **min ↑**

obj = min(obj, outputType, dimension)

Description:

Calculate the min of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their min.

Input:

- obj : An object of class nb_cs
- outputType :
 - > 'double' : Get the min values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the min values as nb_cs object.
- dimension : The dimension to calculate the min over.

Output:

- obj : See the input outputType for more on the output from this method
- dimension : The dimension to calculate the min over.

Examples:

```
double = min(obj,'double');
nb_csObj = min(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **minus ↑**

```
obj = minus(obj,DBOrNum)
```

Description:

Binary subtraction (-).

Caution: Will subtract the data of the corresponding variables from the two objects. All the variables not in common will result in series with nan values.

When one of the inputs are a scalar this function will subtract each element of the data by this number or each element of the data will be subtracted from the scalar.

Input:

- a : An object of class nb_cs or a scalar
- b : An object of class nb_cs or a scalar

Output:

- obj : An nb_cs object where all the variables are from the first object are subtracted from the other.

Or

An nb_cs object where the scalar are substracted from all the elements of the input object

Or

An nb_cs object where all the elements of the input object are substracted from scalar.

Examples:

```
obj = obj - anotherObj;  
obj = obj - 2;  
obj = 2 - obj;
```

See also:

[nb_cs.plus](#), [nb_cs.callop](#), [nb_cs.callfun](#)

Written by Andreas Haga Raavand

► **mldivide** ↑

```
obj = mldivide(a,b)
```

Description:

Matrix left division (\). See examples.

Input:

- a : An object of class nb_cs.
- b : A scalar or a string representing a variable name or type name.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.rand(10,3)
obj = obj\1          % 1/obj
obj = obj\'Type1'   % 'Type1'/obj
obj = obj\'Var1'    % 'Var1'/obj
```

See also:

[nb_cs.ldivide](#), [nb_cs.mrdivide](#), [nb_cs.rdivide](#)

Written by Kenneth S. Paulsen

► **mode** ↑

```
obj = mode(obj,outputType,dimension,varargin)
```

Description:

Calculate the mode of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their mode.

Input:

- obj : An object of class nb_cs
- outputType :
 - > 'double' : Get the mode values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mode values as nb_cs object.
- dimension : The dimension to calculate the mode over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double    = mode(obj,'double');
nb_csObj = mode(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **mpower** ↑

```
obj = mpower(a,b)
```

Description:

Matrix power (^). Defined when raised by a scalar, a string which either represent a variable or type.

Input:

- a : An object of class nb_data
- b : A scalar or a string.

Output:

- obj : An nb_data object.

Examples:

```
obj  = nb_cs.rand(10,2)
obj1 = obj^'Type1'
obj2 = obj^'Var1'
obj3 = obj^1
```

See also:

nb_cs.power

Written by Kenneth S. Paulsen

► **mrddivide** ↑

```
obj = mrdivide(a,b)
```

Description:

Matrix right division (/). See examples.

Input:

- a : An object of class nb_cs, a scalar number or a string representing a variable name or type name.
- b : An object of class nb_cs, a scalar or a string representing a variable name or type name.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.rand(10,3)
obj1 = obj/1
obj2 = obj/'Var1'
obj3 = obj//Type1'
obj4 = 1/obj
obj5 = 'Var1'/obj
obj6 = 'Type1'/obj
```

See also:

[nb_cs.ldivide](#), [nb_cs.mldivide](#), [nb_cs.rdivide](#)

Written by Kenneth S. Paulsen

► **mtimes** ↑

```
obj = mtimes(obj,Num)
```

Description:

Matrix multiplication (*) Defined for multiplication with a constant, a string which either represent a variable or type.

Input:

- obj : An object of class nb_cs or scalar
- b : An object of class nb_cs or a scalar or a string.

Output:

- obj : An nb_cs object where the number (date or variable as a vector) is multiplied with all the elements of the input objects data

Examples:

```
obj = 2*obj;
obj = obj*2;
obj = obj*'Type1'
obj = obj*'Var1'
```

See also:

[nb_cs.times](#)

Written by Kenneth S. Paulsen

► **nan ↑**

```
obj = nb_cs.nan(types,vars)
obj = nb_cs.nan(types,vars,pages)
```

Description:

Create an nb_cs object with all set to nan.

Input:

- types : Either a cellstr with the types of the nb_cs object, or a scalar with the number of types of the nb_cs object.
- vars : Either a cellstr with the variables of the nb_cs object, or a scalar with the number of variables of the nb_cs object.
- pages : Number of pages of the nb_cs object.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.nan({'Type','Type2'},{'Var1','Var2'});
obj = nb_cs.nan(2,2);
obj = nb_cs.nan(2,2,10);
```

See also:

[nb_cs](#)

► **ne** ↑

```
a = ne(a,b)
```

Description:

Test if the one object is not equal to ($\sim=$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **not** ↑

```
a = not(a)
```

Description:

The not operator (~).

Input:

- a : An object of class nb_dataSource.

Output:

- a : An object of class nb_dataSource. Where the not operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = ~a;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► objSize ↑

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim :
 - 1 : Number of observations
 - 2 : Number of variables
 - 3 : Number of datasets (pages)

Output:

- varargout : See the examples below

Examples:

```
dim = objSize(obj)
```

Where dim is 1 x 2 double where the first element is the number of observations/types and the second element is the number of variables

```
[dim1, dim2] = objSize(obj)
```

```
[dim1, dim2, dim3] = objSize(obj)
```

```
dim1 = objSize(obj,1)  
dim2 = objSize(obj,2)  
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **ones** ↑

```
obj = nb_cs.ones(types,vars)
obj = nb_cs.ones(types,vars,pages)
```

Description:

Create an nb_cs object with all set to 1.

Input:

- types : Either a cellstr with the types of the nb_cs object, or a scalar with the number of types of the nb_cs object.
- vars : Either a cellstr with the variables of the nb_cs object, or a scalar with the number of variables of the nb_cs object.
- pages : Number of pages of the nb_cs object.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.ones({'Type','Type2'},{'Var1','Var2'});
obj = nb_cs.ones(2,2);
obj = nb_cs.ones(2,2,10);
```

See also:

[nb_cs](#)

Written by Kenneth Sæterhagen Paulsen

► **or** ↑

```
a = or(a,b)
```

Description:

The and operator (|).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the or operator has evaluated all the data elements of the object. The data will be a logical matrix.

Examples:

```
a = a | b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **packing ↑**

```
obj = packing(obj,packages,varargin)
```

Description:

Package variables by applying the wanted function. Default is summin (sum).

Input:

- obj : An object of class nb_ts.
- packages : A 2 x N cell matrix with groups of variables and the names of the groups. If you have not listed a variable it will be grouped in a group called 'Rest' (as long as 'includeRest' is not set to false).

Must be given in the following format:

```
{'group_name_1','group_name_N';
 name_1 ,...,name_N}
```

Where name_x must be a string with a name of the variable or a cellstr array with the variable names. E.g 'Var1' or {'Var1','Var2'}.

Optional inputs:

- 'includeRest' : Give false to not include the 'Rest' variable, which will represent all the variable that has not been packed.
- 'func' : Either a function_handle or a one line char with the

name of the function to use. The first input to this function is a nObs x nVar double, the second is the dimension (which is always 2). Examples; 'sum', 'prod', @sum or @prod.

Output:

- obj : A object of class nb_ts.

Examples:

```
data      = nb_ts.rand('2012Q1',10,4);
packages = {'group1','group2';
            {'Var1','Var2'},{'Var3'}};

dataP1 = packing(data,packages)
dataP2 = packing(data,packages,'includeRest',false)
dataP3 = packing(data,packages,'func',@prod)
```

See also:

[nb_ts.createVariable](#), [nb_data.createVariable](#), [nb_cs.createVariable](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **percentiles** ↑

obj = percentiles(obj,perc)

Description:

Take the percentile over the third dimension of the data of the object.

Input:

- obj : An object of class nb_cs
- perc : A double with the wanted percentiles of the data. E.g. 0.5 (median), or [0.05,0.15,0.25,0.35,0.5,0.65,0.75,0.85,0.95].

Output:

- obj : A nb_cs object with the percentiles stored as different pages of the object. See the dataNames property for the ordering.

Written by Kenneth Sæterhagen Paulsen

► **permute** ↑

```
obj = permute(obj,variables)
```

Description:

Switch the second and third dimension of the nb_ts, nb_cs or nb_data object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- variables : A cellstr with size 1 x numberOfDatasets. Default is to use dataNames property. I.e. when not provided or if empty.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **plot ↑**

```
plot(obj,varargin)
plotter = plot(obj,varargin)
```

Description:

Plot the data of the nb_dataSource object.

Caution: If the data of the object is of class nb_distribution this method will redirect to plotDist.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

- plotType : A string with either:

```
> 'graph'    (Default)  
> 'graphSubPlots'  
> 'graphInfoStruct'
```

The name of the graphing method of the class nb_graph_ts, nb_graph_cs or nb_graph_data.

Caution: This is an optional input. If not given it is assumed that the optional input pairs are starting from the second input to this function.

```
- varargin : Same as the input to the method set() of the
            nb_graph_ts, nb_graph_cs or nb_graph_data class.

            I.e. ..., 'inputName', inputValue, ...

Caution: If the data of the object is of class
         nb_distribution, see nb_ts.plotDist for the
         optional input pairs supported.
```

Output:

Plotted output

Examples:

```
plot(obj)

plot(obj,'graphSubPlots')

plot(obj,'','startGraph','2008Q1','endGraph','2011Q2')

same as

plot(obj,'graph','startGraph','2008Q1','endGraph','2011Q2')
```

See also:

[nb_graph_ts](#), [nb_ts.plotDist](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **plus ↑**

```
obj = plus(obj,DBOrNum)
```

Description:

Binary addition (+).

Caution: Will add the data of the corresponding variables from
 the two objects. All the variables not in common will result
 in series with nan values.

When one of the inputs are a scalar this function will add
 each element of the data by this number.

Input:

- a : An object of class nb_cs or a scalar
- b : An object of class nb_cs or a scalar

Output:

- obj : An nb_cs object where all the variables from the two objects are added.

Or

An nb_cs object where the scalar are added to all elements of the given nb_cs object

Examples:

```
obj = obj + anotherObj;  
obj = obj + 2;  
obj = 2 + obj;
```

See also:

[nb_cs.minus](#), [nb_cs.callop](#), [nb_cs.callfun](#)

Written by Andreas Haga Raavand

► **pow2** ↑

```
obj = pow2(obj)
```

Description:

`pow2(obj)` raises the number 2 to the power of the elements of `obj`

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **power** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^).

If both inputs are nb_cs objects this method will raise the data of the first object with the data from the second object. This will only be done for the variables which are found to be in both nb_cs objects. The rest of the variables will get nan values.

Input:

- obj : An object of class nb_cs or a scalar
- DBOrNum : An object of class nb_cs or a scalar

Output:

- obj : An nb_cs object where the data from th one object are raised with the data from the other object

or

An nb_cs object where all the elements are raised by the scalar DBOrNum.

or

An nb_cs object where the scalar obj are raised by all the elements of the object DBOrNum.

Examples:

```
obj = a.^b;
obj = a.^2;
obj = 2.^a;
```

See also:

[nb_cs.mpower](#), [nb_cs.callop](#), [nb_cs.callfun](#)

Written by Andreas Haga Raavand

► rand ↑

```
obj = nb_cs.rand(types,vars)
obj = nb_cs.rand(types,vars,pages)
```

Description:

Create a random nb_cs object

Input:

- types : Either a cellstr with the types of the random nb_cs object, or a scalar with the number of types of the random nb_cs object.
- vars : Either a cellstr with the variables of the random nb_cs object, or a scalar with the number of variables of the random nb_cs object.
- pages : Number of pages of the random nb_cs object.
- dist : A nb_distribution object to draw from.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.rand({'Type','Type2'},{'Var1','Var2'});
obj = nb_cs.rand(2,2);
obj = nb_cs.rand(2,2,10,nb_distribution('type','normal'));
```

See also:

[nb_cs](#)

Written by Kenneth Sæterhagen Paulsen

► [rdivide](#) ↑

```
obj = rdivide(obj,DB)
```

Description:

Right element-wise division (./)

Do right element-wise division of the matching variables of the two nb_cs objects. If some variables are not found to be in both objects, the resulting object will have series with nan values only for these variables.

Input:

- a : An object of class nb_cs
- b : An object of class nb_cs

Output:

- obj : An object of class nb_cs, where the data are the result of the element-wise division.

Examples:

```
obj = obj./DB;
```

See also:

[nb_cs.mldivide](#), [nb_cs.mrdivide](#), [nb_cs.callop](#), [nb_cs.callfun](#)

Written by Kenneth S. Paulsen

► **real** ↑

```
obj = real(obj)
```

Description:

`real(obj)` returns the real part of the elements in the `nb_cs` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **reallog** ↑

```
obj = reallog(obj)
```

Description:

Take the real natural logarithm of the data stored in the object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = reallog(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **realpow** ↑

obj = realpow(obj,DBOrNum)

Description:

Same as nb_cs.power.

See also:

[nb_cs.power](#)

Written by Kenneth S. Paulsen

► **realsqrt** ↑

obj = realsqrt(obj)

Description:

realsqrt(obj) is the square root of the elements of obj.
An error is produced if obj contains negative elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = realsqrt(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► rename ↑

```
obj = rename(obj,type,oldName,newName)
```

Description:

Makes it possible to rename variables and datasets. For objects of class nb_cs types is also possible to rename.

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variable', but not when the input is 'dataset' or 'types'.

Input:

- obj : An object of class nb_dataSource
- type : Either:
 - 'variable' : To rename a variable(s)
 - 'dataset' : To rename a dataset(s) (The only input that is supported for nb_cell)
 - 'types' : To rename a type(s) (only nb_cs)
- oldName : Case 1;
 - > The old variable/dataset/type name, as a string
- Case 2:
 - > The string which should be replaced in all the variable names of the database. And reorder things afterwards. This input must end with a *.
- newName : Case 1;
 - > The new variable/dataset/type name, as a string
- Case 2;
 - > The string which should replace the old string part given by the input 'oldName'.

Output:

- obj : An nb_dataSource object with the renamed variable(s), type(s) or dataset(s).

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.rename('variable','Var1','Var3');  
obj = obj.rename('variable','Var*','N_Var');  
obj = rename(obj,'variable',{'Var1','Var2'},{'VAR1','VAR2'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **renameMore** ↑

```
obj = renameMore(obj,type,oldNames,newNames)
```

Description:

Makes it possible to rename more variables, datasets and types (only nb_cs).

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variables', but not when the input is 'datasets'.

Caution : This is the same as looping over nb_dataSource.rename, but more efficient.

Caution : * syntax is not supported as is the case for nb_dataSource.rename

Input:

- obj : An object of class nb_dataSource.
- type : Either:
 - 'variables' : To rename a variables
 - 'datasets' : To rename a datasets (The only input that is supported for nb_cell)
 - 'types' : To rename a type (Only nb_cs)
- oldNames : The old variable/dataset/type names, as a cellstr.
- newName : The new variable/dataset/type names, as a cellstr.

Output:

- obj : An nb_dataSource object where the variable(s)/dataname(s)/type(s) of the object given as input is/are renamed.

Examples:

```
obj = nb_cs([22,24;27,28],'Dataset1',{'Exp','Imp'},{'Nor','Swe'})  
obj = renameMore(obj,'dataset',{'Dataset1'},{'tradeBal'});  
obj = renameMore(obj,'variable',{'Nor','Swe'},{'Norway','Sweden'});  
obj = renameMore(obj,'type',{'Exp','Imp'},{'Export','Import'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **reorder** ↑

```
obj = reorder(obj,newOrder,type)
```

Description:

Re-order the variables/types/datasets of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- newOrder : A cellstr with the new order of the variables/types/datasets.
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **round** ↑

```
obj = round(obj,value,types,variables)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_cs object
- value : The rounding value. As a double. Default is 1.
- types : The types to round. Can be empty. Default is to round all types.
- variables : The variables to round. Can be empty. Default is to round all variables.
- pages : A numerical index of the pages you want to keep.

Output:

- obj : A nb_cs object with the rounded numbers.

Written by Kenneth Sæterhagen Paulsen

► **saveDataBase** ↑

```
saveDataBase(obj,varargin)
```

Description:

Save data of the nb_cs object

If no optional input is given the data of the object is save down to xlsx file(s). Each dataset of the object is saved down to a excel spreadsheet with the name of the dataset (Taken from the dataNames property of the object)

Input:

- obj : An object of class nb_cs

Optional input (...'propertyName',PropertyValue,...):

- varargin :

> 'saveName' : The data is saved to a excel worksheet with the same name as the input after 'saveName'. If consisting of more dataset, each page of the data property is saved with the same name as the input after 'saveName' pluss the page number.

> 'ext' :

> 'xlsx' : Default. Save data to excel spreadsheet(s)

> 'matold' : Save the data down to a mat file. The data is saved down as a structure, with the variables as the fieldnames, and the data as its fields. If the data consist of more pages, each field will consist of the same number of pages.

The types of the object is also added to the field 'types' of the saved structure. Which makes it possible to load the .mat field up again to nb_cs object.

See the toStructure method with input old set to 1.

Must be combined with the optional input 'saveName'.

> 'mat' : Save the data down to a mat file. The data is saved down as a structure, with the variables stored in the field 'variables', while the data of the variables will be saved as its fields with generic names ('Var1','Var2' etc). If the data consist of more pages, each field will consist of the same number of pages.

The types of the object is also added to the field 'types' of the saved structure. Which makes it possible to load the .mat field up again to nb_cs object.

See the `toStructure` method with input `old` set to 1.

Must be combined with the optional input
`'saveName'`.

> `'mats'` : The object is saved down as a struct using
the `nb_cs.struct` function.

> `'txt'` : Save the data to .txt file.

> `'append'` :

> 1 : Saves all the datasets (pages of the data) to one
excel spreadsheet, but in seperate worksheets.
(With the names of the dataset, given by `dataNames`
property of the object.)

Works only when `'ext'` input is set as `'xlsx'`. (The
default)

Caution : If saved in this way, it is not possible
to load it up again to a `nb_cs` object

> 0 : The default saving method.

> `'sheets'` : A cellstr with the sheet names to save the datasets/pages
of the object down to. Must match the `dataNames` property.
This option has precedence over `'append'`.

Output:

Saved output in the wanted format.

Examples:

Save data to excel:

```
obj = nb_cs(ones(1,2,2)*2,'test',{'First'},{'Var1','Var2'});
obj.saveDataBase();
obj.saveDataBase('saveName','test','append',1);
```

Save data to .mat file:

```
obj.savDataBase('ext','mat','saveName','test')
```

See also:

[asCell](#), [toStructure](#), [nb_readMat](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

► **sec ↑**

```
obj = sec(obj)
```

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **secd** ↑

```
obj = secd(obj)
```

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **setInfinite2NaN** ↑

```
obj = setInfinite2NaN(obj)
```

Description:

Set all elements that are isfinite to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setMethodCalls** ↑

```
obj = setMethodCalls(obj,c)
```

Description:

Set all method calls of the object with a cell matrix. The object needs to be updatable.

Caution: To delete method calls use deleteMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.
- c : A cell matrix. See the output of getMethodCalls

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

See also:

[nb_cs.deleteMethodCalls](#), [nb_cs.getMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► setValue ↑

obj = setValue(obj,VarName,Value,Types,pages)

Description:

Set some values of the dataset(s). (Only one variable.)

Input:

- obj : An object of class nb_cs
- VarName : The variable name of the variable you want to set some values of. As a string
- Value : The values you want to assign to a variable. Must be a numerical vector (with the same number of pages as given by pages or as the number of dataset the object consists of.) Must have the same length as the input given by Types. (If this is not provided this input must of same length as the number of existing types of the object)
- Types : A cellstr of the type names to set.
- pages : At which pages you want to set the values of a variable. Must be a numerical index of the pages you want to set.
E.g. if you want to set the values of the 3 first datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages must be set.

Output:

- obj : An nb_cs object with the added variable

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
obj = obj.setValue('Var2',3);
obj = obj.setValue('Var2',4,['First'],1);
```

Written by Kenneth S. Paulsen

► **setZero2NaN** ↑

```
obj = setZero2NaN(obj)
```

Description:

Set all elements that are 0 to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sin** ↑

```
obj = sin(obj)
```

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sin(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_DATASOURCE
```

► **sind** ↑

```
obj = sind(obj)
```

Description:

sind(obj) is the sine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

sinh(obj) is the hyperbolic sine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sinh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE
```

► **skewness** ↑

```
obj = skewness(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the skewness of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their skewness.

Input:

- obj : An object of class nb_cs
- flag : Same as for the function skewness created by MATLAB.
- outputType :
 - > 'double' : Get the skewness values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the skewness values as nb_cs object.
- dimension : The dimension to calculate the skewness over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method
- dimension : The dimension to calculate the skewness over.

Examples:

```
double    = skewness(obj,0,'double');
nb_csObj = skewness(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► **sort** ↑

```
obj = sort(obj,dim,mode,variable)
```

Description:

Sort the nb_cs object in the given dimension

Caution : Sorting does not reorder the dataNames, variables or types!

Input:

- obj : An object of class nb_cs
- dim : The dimension to sort
- mode : 'descend' (default)
 'ascend'
- variable : Give a string with the variable to sort. The rest of the variables will then be reordered accordingly.

 If empty (default), all variables are sorted.

Caution: Only an option for dim == 1

Output:

- obj : An nb_cs object representing the sorted input

Examples:

```
obj = nb_cs([1,2;1 1],'test',{'First','Second'},{'Var1','Var2'});  
obj = sort(obj,1,'ascend')
```

Written by Kenneth S. Paulsen

► **sortProperty** ↑

```
obj = sortProperty(obj,type)
```

Description:

Sort the variables/types/datasets of the object alphabetically.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► sortTypes ↑

```
obj = sortTypes(obj,typesOrder)
```

Description:

Sort the types alphabetically, or reorder them according to typesOrder.

Input:

- obj : An object of class nb_cs
- typesOrder : A cellstr with the wanted order of the types. If empty an alphabetical order is used.

Caution: Must contain all types of the object if not empty.

Output:

- obj : An object of class nb_cs.

Written by Kenneth Sæterhagen Paulsen

► sqrt ↑

```
obj = sqrt(obj)
```

Description:

`sqrt(obj)` is the square root of the elements of `obj`. Complex results are produced for non-positive elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **squeeze ↑**

```
obj = squeeze(obj)
```

Description:

Squeezes the different datasets (pages) of the given nb_cs object. It will append the variables in each dataset with the given data name

Input:

- obj : An nb_cs object. If the object has only one dataset (page) this method has nothing to do.

Output:

- obj : An nb_cs object, where all the pages are squeezed together. I.e. all data will be stored in one dataset, and all variable names will get the dataset name appended.

Example:

```
obj = nb_cs(ones(1,1,2),{'b','c'},{'Type'},{'Var1'})
```

```
obj =
```

```
(:,:,1) =
```

```
'Types'    'Var1'  
'Type'     [ 1]
```

```
(:,:,2) =
```

```
'Types'    'Var1'  
'Type'     [ 1]
```

```
sObj = squeeze(obj)
```

```
sObj =
```

```
'Time'    'Var1_b'    'Var1_c'  
'2012'   [ 1]      [ 1]
```

Written by Kenneth Sæterhagen Paulsen

► std ↑

```
obj = std(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the std of each series. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their std.

The output can also be set to be a double.

Input:

- obj : An object of class nb_cs
 - flag :
 - > 0 : normalises by N-1 (Default)
 - > 1 : normalises by N
- Where N is the sample length.
- outputType :
 - > 'double' : Get the std values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the std values as nb_cs object.
 - dimension : The dimension to calculate the std over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
double = std(obj,0,'double');
nb_csObj = std(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► stopSorting ↑

```
obj = stopSorting(obj)
```

Description:

Stop sorting of variables.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where sorting is turned off.

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **struct** ↑

```
s = struct(obj)
```

Description:

Object to struct.

Input:

- obj : An nb_cs object.

Output:

- s : A structure representing an nb_cs object.

Written by Kenneth Sæterhagen Paulsen

► **structure2Dataset** ↑

```
obj = structure2Dataset(obj,givenStruct,NameOfDataset)
```

Description:

When you give a struct to the constructor of this class all the fields af that struct will be added as a new dataset, but this method add all the fields of the input givenStruct as a variable in one dataset

If you only want to add a dataset from a structure in this way (no other datasets) initialize a empty object, and then use this method.

Caution : The structure must include a field 'types', which must be a cellstr with the added types of the nb_cs object. This cellstr must have the same number length as the length of the data of the other field

Inputs:

- obj : The object itself, could be empty.
- givenStruct : The structure which contains the data of the added variable. Where the fieldnames will be the variables name of the added dataset + plus one field 'types' with the data types of the dataset.
- NameOfDataset : Name of the added dataset. If not provided default name will be used, i.e Database<jj>

Outputs:

- obj : An nb_cs object with the added dataset.

Examples:

```
s      = struct();
s.types = {'First'};
s.var1  = 2;
obj    = nb_cs;
obj    = obj.structure2Dataset(s)

obj =
{'Types'   'var1'
 'First'   [ 2]
```

Written by Kenneth S. Paulsen

► **subsasgn** ↑

```
varargout = subsasgn(obj,s,b)
```

Description:

Makes it possible to do subscripted assignment of the data of an nb_cs object

Input:

- obj : AN object of class nb_cs
- s : The same input as when you do subscripted assignment
one a double matrix
- b : The assign data at the given index s.

Output:

- obj : An nb_cs object where the assign data property is setted.

Examples:

```
obj(1:2) = [2;3];
```

same as

```
obj(1:2,1,1) = [2;3];
```

same as

```
obj.data(1:2,1,1) = [2;3];
```

```
obj(1:2,2,3) = [2;3];
```

Written by Kenneth S. Paulsen

► subsref ↑**Description:**

Makes it possible to do subscripted reference of the data of an nb_cs object. See the examples for more

Uses the method window, just using indexing instead

Input:

- obj : An object of class nb_cs
- s : A structure on have to index the object

Output:

- obj : An nb_cs object indexed as wanted.

Examples:

```

obj = obj(1,2:4,1)

same as

obj = obj(1,2:4)

obj = obj({'Type1','Type2',...})

same as

obj.window({'Type1','Type2',...})

obj = obj({'Type1','Type2',...},{'Var1','Var2',...})

same as

obj.window({'Type1','Type2',...},{'Var1','Var2',...})

obj = obj({'Type1','Type2',...},{'Var1','Var2',...},1:2)

same as

obj.window({'Type1','Type2',...},{'Var1','Var2',...},1:2)

obj = obj('Var1','Var2',...)

same as

obj.window('',',{Var1','Var2',...},1:obj.numberOfDatasets)

```

Written by Kenneth S. Paulsen

► sum ↑

```
obj = sum(obj,dim)
```

Description:

Take the sum over a given dimension of an nb_cs object

Input:

- obj : An object of class nb_cs
- dim : The dimension to take the sum over

Output:

- obj : An nb_cs object representing the sum

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
obj = sum(obj,2)
obj =
'Types'      'sum'
'First'       [ 4]
```

Written by Kenneth S. Paulsen

► **summary** ↑

```
obj = summary(obj)
```

Description:

Summary of the series of the nb_cs object

Input:

- obj : An object of class nb_cs

Output:

- obj : the summary statistics - min, max, std, var, skewness and kurtosis

Written by Kenneth S. Paulsen

► **tail** ↑

```
tail(obj,nRows,page)
```

Description:

Display the last n rows of a nb_cs object.

Input:

- obj : An nb_cs object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

`nb_cs, window, head`

Written by Per Bjarne Bye

► **tan ↑**

`obj = tan(obj)`

Description:

Take the tangent of the data stored in the `nb_ts`, `nb_cs` or `nb_data` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

`obj = tan(obj);`

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **tand ↑**

`obj = tand(obj)`

Description:

`tand(obj)` is the tangent of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = tand(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

`tanh(obj)` is the hyperbolic tangent of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = tanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **times** ↑

```
obj = times(obj, DB)
```

Description:

Element-wise multiplication (`.*`)

Caution : Will only multiply the data of corresponding variables of the two objects. I.e. variable not found to be in both object will result in series with all values set to nan.

Input:

- `obj` : An object of class `nb_cs`
- `DB` : An object of class `nb_cs`

Output:

- obj : An nb_data object the data results from element-wise multiplication of the data of the input objects.

Examples:

```
obj = obj.*DB;
```

See also:

[nb_cs.mtimes](#), [nb_data.callop](#), [nb_data.callfun](#)

Written by Andreas Haga Raavand

► **toMFile** ↑

```
string = toMFile(obj)
toMFile(obj,filename)
string = toMFile(obj,filename,varName)
```

Description:

Write a .m file to generate the current object.

Input:

- obj : An nb_cs object.
- filename : The filename to write the code to. If empty no file will be written.
- varName : Name of decleared variable by this code. Default is data.

Output:

- string : A cellstr with the .m code to generate the current object.

Written by Kenneth Sæterhagen Paulsen

► **toStructure** ↑

```
retStruct = toStructure(obj)
retStruct = toStructure(obj,old)
```

Description:

Transform the obj to a structure

Input:

- obj : An object of class nb_cs
- old : Indicate if old mat file format is wanted. Default is 0 (false).

Output:

- retStruct : A structure with fieldsnames given by the object variables and field given as the variables data. (Can have more than one page).

Caution: A extra field is also added to save the objects types. (Saved under the field 'types')

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
s   = obj.toStructure()
s =
    Var1: 2
    Var2: 2
    variables: {'Var1'  'Var2'}
    types: {'First'}
    class: 'nb_cs'
    userData: ''
    localVariables: []
    sorted: 1
```

See also:

[nb_cs.struct](#)

Written by Kenneth S. Paulsen

► **tonb_cell ↑**

nb_cell_DB = tonb_cell(obj)

Description:

Transform from a nb_cs object to a nb_cell object

Input:

- obj : An object of class nb_cs

Output:

- nb_cell_DB : An object of class nb_cell

Examples:

```
nb_cell_DB = obj.tonb_cell();
```

Written by Kenneth S. Paulsen

► **tonb_data ↑**

```
nb_data_DB = tonb_data(obj)
```

Description:

Transform from an nb_cs object to an nb_data object

Caution : This will only succeed if the types can be interpreted as observations.

Input:

- obj : An object of class nb_cs

Output:

- nb_data_DB : An object of class nb_data

Examples:

```
nb_data_DB = obj.tonb_data();
```

Written by Kenneth S. Paulsen

► **tonb_ts ↑**

```
nb_ts_DB = tonb_data(obj)
```

Description:

Transform from a nb_cs object to a nb_ts object

Caution : This will only succeed if the types can be interpreted as dates.

Input:

- obj : An object of class nb_cs

Output:

- nb_data_DB : An object of class nb_ts

Examples:

```
nb_ts_DB = obj.tonb_ts();
```

Written by Kenneth S. Paulsen

► **transpose ↑**

```
obj = transpose(obj)
```

Description:

Take the transpose (.)' of an nb_cs object.

Caution : The former types will be sorted when transposing the object.

Input:

- obj : An object of class nb_cs

Output:

- obj : An nb_cs object which represents the transposed input object

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'})
```

```
obj =
```

```
'Types'    'Var1'    'Var2'  
'First'     [ 2]     [ 2]
```

```
t = obj.'
```

```
t =
```

```
'Types'    'First'  
'Var1'      [ 2]  
'Var2'      [ 2]
```

Written by Kenneth S. Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : A nb_ts, nb_cs or nb_data object where all the data are the unary minus of the input objects data

Examples:

```
obj = -obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **unstruct** ↑

```
obj = nb_cs.unstruct(s)
```

Description:

Reverse of the struct method.

Input:

- s : See the s output of the struct method.

Output:

- obj : An nb_cs object.

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,warningOff,inGUI)
```

Description:

This method will try to update the data of the nb_ts object.

Caution : It is only possible to update an nb_dataSource object which has updateable links to a FAME database or a full directory (path) to an excel spreadsheet or .mat file or the SMART database.

Caution : If you want to create a link to a specific worksheet of a excel file you must provide the extension!

Caution : All method calls done on the object are preserved.
(There exist some exception; and, or etc.)

Input:

- obj : An object of class nb_dataSource which are updatable.
- warningOff : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.

Output:

- obj : An object of class nb_dataSource with the data updated. If it is not possible a warning will be given.

Examples:

```
obj = nb_ts(['N:\Matlab\Utvikling\MATLAB_LIB\NEMO\'...
             'Documentation\Examples\example_ts_quarterly']);
obj = obj.update
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► uplus ↑

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► var ↑

```
obj = var(obj,outputType,dimension,varargin)
```

Description:

Calculate the var of object. The result will be an object of class nb_cs where all the non-nan values of all the object are being set to their var.

The output can also be set to be a double or a nb_cs object consisting of the var values of the data.

Input:

- obj : An object of class nb_cs
- outputType :
 - > 'double' : Get the var values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the var values as nb_cs object.
- dimension : The dimension to calculate the var over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
double    = var(obj,'double');
nb_csObj = var(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **vertcat** ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_cs
- b : An object of class nb_cs
- varargin : Optional numbers of objects of class nb_cs

Output:

- a : An nb_cs object where the data from the different objects are appended to each other.

Examples:

Written by Kenneth S. Paulsen

► **window** ↑

```
obj = window(obj,typesWin,variablesWin,pages)
```

Description:

Narrow down the dataset(s) of the nb_cs object

Input:

- obj : An object of class nb_cs
- typesWin : A cellstr array of the types you want to keep of the given object. If empty all the types is kept.
- variablesWin : A cellstr array of the variables you want to keep of the given object. If empty all the

variables is kept.

- pages : A numerical index of the pages you want to keep.
E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages is kept.

I.e. `obj = obj.window('','','',1:3)`

Or it can be the name of the dataset you want to keep. Then the input must be a string.

I.e. `obj = obj.window('','','','Database1')`

Or it can be a cell array of the datasets names to keep.

I.e `obj = obj.window('','','',{ 'Database1',...,'DataBase2' })`

Output:

- `obj` : An `nb_cs` object where the datasets of the input object are narrowed down to the given window.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.window({},{'Var1'},1);
```

Written by Kenneth S. Paulsen

► **writeTex** ↑

```
writeTex(obj,filename)  
writeTex(obj,filename,'inputName', inputValue,...)
```

Description:

Writes the object data to a LaTeX table saved to a `.tex` file.

If the object consists of more pages (datasets) more tables will be created.

Input:

- `obj` : An object of class `nb_cs`
- `filename` : The name of the saved `.tex` file. With or without the extension.

Optional input:

See optional inputs to nb_cs.getCode.

Output:

A saved .tex file with the LaTeX table

Examples:

```
obj = nb_cs([2,2,2],'',{'type1'},{'Var1','Var2','Var3'});
obj.writeTex('test')
```

See also:

[nb_cs](#), [nb_cs.getCode](#)

Written by Kenneth SÅ!terhagen Paulsen

► **zeros** ↑

```
obj = nb_cs.zeros()
obj = nb_cs.zeros(types,vars,pages)
```

Description:

Create an nb_cs object with all set to 0.

Input:

- types : Either a cellstr with the types of the nb_cs object, or a scalar with the number of types of the nb_cs object.
- vars : Either a cellstr with the variables of the nb_cs object, or a scalar with the number of variables of the nb_cs object.
- pages : Number of pages of the nb_cs object.

Output:

- obj : An nb_cs object.

Examples:

```
obj = nb_cs.zeros({'Type','Type2'},{'Var1','Var2'});
obj = nb_cs.zeros(2,2);
obj = nb_cs.zeros(2,2,10);
```

See also:

■ nb_data

Go to: [Properties](#) | [Methods](#)

A class representing data with links to the variable names.

Constructor:

```
obj = nb_data(datasets,NameOfDatasets,startObs,variables)
obj = nb_data(datasets,NameOfDatasets,startObs,variables,sorted)
```

Input:

- datasets :

Input can be one of the data types listed below:

- xls(x) : A name of the excel spreadsheet to import.
(With or without extension.)

Caution : If you read a specific worksheet, and
want the object to be updateable
you must provide the extension.

Caution : The first column of the worksheet must start
with 'obs' or 'observations', and the rest
of the column you must provide the
observation number to each row. E.g:

'obs'		'Var1'

1		2.5

2		2.0

- mat : Name(s) of the .mat files to import. (With or
without extension.).

Format of the .mat file:

The .mat file must be saved structure with the
fieldnames as the variable names and the fields
as the data of the variables, as doubles. Plus
it must have a field with name 'startObs' with
the start obs of the data, as an integer.

- double : As double matrix. Each page of the matrix will
be added as separate datasets.

- struct : As a structure of either double matrices. Each field is
added as separate datasets.

You can also give a cell array consisting of one or a
combination of the data types mentioned above. Each cell

will be added as a new dataset.

Extra: Use the method structure2nb_data function to add all the fields of the structure as variables in an nb_data object. (If each field has more page then 1, then each page will be a datasets)

- NameOfDatasets :

Input options will depend on data types listed below:

- xls(x) : A string with the wanted dataset name. Default is the excel file name. If the provided string is a sheet of the read spreadsheet that specific worksheet will be read.

- mat : A string with the wanted dataset name. Default is the .mat file name.

- double : A string with the wanted dataset name. Default is 'Dataset1'.

Caution: When you given double matrices which has more pages then one, either directly or through a struct. Each of the pages will be added as a dataset. The name of this datasets will, if the input NameOfDatasets has the same size as the input datasets, get the name NameOfDatasets{jj}<jj>, where jj is the page number, or else get the name 'Database<jj>'.

- struct : This input has nothing to say. The dataset names will be given by the fieldnames.

- A cell array of the listed types above:

If you give a cell array to the datasets input, this input must be a cell array of strings. If you give a cell array which doesn't have the same size as the cell array given to the datasets input, the datasets which is left gets the name 'Database<jj>', where <jj> stands for this added datasets number.

If you give an empty cell array, i.e. {} all the datasets get the names; 'Database<jj>', where <jj> stands for the added datasets number. Except when the datasets input are given by a cellstr with the excel file names, .mat file names or a FAME path names, then the default dataset names will be set to the same cellstr. Or if some elements of the cell is a struct, then the datasets provided through the struct will get their fieldnames as the dataset names.

- startDate :

The start date of the data. Only needed when you give numerical data as one element of the input datasets (also when numerical data is given through a struct). Default is 1.

```

- variables      :

A cellstr array of the names of the variables.

Needed when the dataset input is:

> Numerical matrix. The number of variables must be of the
same as the size of the data (2. dimension).

> A struct consisting of double matrices.

> When you give a cell array of one of the types mentioned
above.

- sorted        : true or false. Default is true.

Output:

- obj          : An object of class nb_data

Examples:

- xls(x) :

    One excel spreadsheet

    obj = nb_data('test1')

    More excel spreadsheets

    obj = nb_data({'test1','test2',...})

    same as

    obj = nb_data({'test1','test2',...},{'test1','test2',...})

- mat:

    One .mat file

    obj = nb_data('test1')

    More .mat files

    obj = nb_data({'test1','test2',...})

    same as

    obj = nb_data({'test1','test2',...},{'test1','test2',...})

-struct:

    One struct

    > Consisting of double matrices:

    obj = nb_data(struct1,{},1,['Var1','Var2',...])

```

More structs

> Consisting of double matrices:

```
obj = nb_data({struct1,struct2,...},[],1,...  
{'Var1','Var2',...})
```

- double matrix:

One matrix

```
obj = nb_data(double1,'',1,['Var1','Var2',...])
```

```
obj = nb_data(double1,'test1',1,['Var1','Var2',...])
```

More matrices

```
obj = nb_data({double1,double2,...},[],1,...  
{'Var1','Var2',...})
```

```
obj = nb_data({double1,double2,...},{'name1','name2',...},...  
1,['Var1','Var2',...])
```

One matrix with more pages:

```
obj = nb_data(double1,[],1,['Var1','Var2',...])
```

```
obj = nb_data(double1,{'name1','name2',...},1,...  
['Var1','Var2',...])
```

See also:

[nb_cs](#), [nb_ts](#), [nb_graph_ts](#), [nb_graph_cs](#), [nb_graph_data](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [data](#)
- [dataNames](#)
- [endObs](#)
- [localVariables](#)
- [numberOfDatasets](#)
- [numberOfObservations](#)
- [numberOfVariables](#)
- [startObs](#)
- [userData](#)
- [variables](#)

- **data** ↑

The data of the object. As a double or logical.

Inherited from superclass NB_DATASOURCE

- **dataNames** ↑

The names of the different dataset. As a cellstr.

Inherited from superclass NB_DATASOURCE

- **endObs** ↑

The end obs of the data. As an integer.

- **localVariables** ↑

A nb_struct with the local variables of the nb_ts object.
I.e. a field with name 'test' can be reach with the string
input %#test. Only for dates and vintage inputs.

Inherited from superclass NB_DATASOURCE

- **numberOfDatasets** ↑

Number of datasets stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **numberOfObservations** ↑

Number of observation of the data stored
in the object. As an integer.

- **numberOfVariables** ↑

Number of variables stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **startObs** ↑

The start obs of the data. As an integer.

- **userData** ↑

Add user data. Can be of any type.

Inherited from superclass NB_DATASOURCE

- **variables** ↑

Variables of the object. As a cellstr.

Inherited from superclass NB_DATASOURCE

Methods:

- abs
- acosh
- acoth
- acsch
- addPageCopies
- addPrefix
- append
- asCellForMPRReport
- asech
- asinh
- atand
- bkfilter
- callfun
- ceil
- cos
- cot
- cov
- createVariable
- csch
- cumsum
- deleteVariables
- detrend
- double
- empty
- estimateDist
- expandPeriods
- floor
- acos
- acot
- acsc
- addDataset
- addPages
- addVariable
- appendFromAnotherPage
- asec
- asin
- assignNaN
- atanh
- bkfilter1s
- callop
- conj
- cosd
- cotd
- createObsDummy
- csc
- cumnansum
- cutAtObs
- demean
- diff
- ecdf
- epcn
- exp
- expm1
- fromRise_ts
- acosd
- acotd
- acscd
- addDatasets
- addPostfix
- and
- asCell
- asecd
- asind
- atan
- autocorr
- breakLink
- callstat
- corr
- cosh
- coth
- createVarDummy
- cscd
- cumprod
- deleteMethodCalls
- deptcat
- disp
- egrowth
- eq
- expand
- extMethod
- ge

- get
- getClassOfData
- getCode
- getCreatedVariables
- getMethodCalls
- getRealEndObs
- getRealStartObs
- getSource
- getSourceList
- getVariable
- growth
- gt
- hasVariable
- hdi
- head
- histcounts
- histogram
- horzcat
- hpfilter
- hpfilter1s
- iegrowth
- iepcn
- igrowth
- interpolate
- ipcн
- isBoolean
- isCrossSectional
- isDistribution
- isDouble
- isTimeseries
- isUpdateable
- isempty
- isfinite
- isnan
- isscalar
- keepPages
- keepVariables
- ksdensity
- kurtosis
- lag
- lastObservation
- le
- lead
- log
- log10
- log1p
- log2
- lt
- map2Distribution
- mavg
- max
- mean
- meanDiff
- meanGrowth
- median
- merge
- merge2Series
- min
- minus
- mldivide
- mode
- mpower
- mrdivide
- mstd
- mtimes
- nan
- nan2var
- ne
- not
- objSize
- observations
- ones
- or
- packing
- parcorr
- pca
- pcn
- percentiles
- permute
- plot
- plus

- pow2
 - rdivide
 - reallog
 - rename
 - ret
 - saveDataBase
 - sech
 - setMethodCalls
 - sin
 - skewness
 - splitSeries
 - std
 - strip
 - structure2Dataset
 - sum
 - tan
 - times
 - tonb_cell
 - uminus
 - update
 - vertcat
 - zeros
 - power
 - reIndex
 - realpow
 - renameMore
 - rlag
 - sec
 - set2nan
 - setValue
 - sind
 - sort
 - sqrt
 - stdise
 - stripButLast
 - subsasgn
 - summary
 - tand
 - toMFfile
 - tonb_cs
 - undiff
 - uplus
 - window
 - rand
 - real
 - realsqrt
 - reorder
 - round
 - secd
 - setInfinite2NaN
 - setZero2NaN
 - sinh
 - sortProperty
 - squeeze
 - stopSorting
 - struct
 - subsref
 - tail
 - tanh
 - toStructure
 - tonb_ts
 - unstruct
 - var
 - writeTex
-

► **abs** ↑

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acos** ↑

```
obj = acos(obj)
```

Description:

Take acos of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = acos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acosd** ↑

```
obj = acosd(obj)
```

Description:

acosd(obj) is the inverse cosine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acosd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acosh** ↑

```
obj = acosh(obj)
```

Description:

acosh(obj) is the inverse hyperbolic cosine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acosh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acot** ↑

```
obj = acot(obj)
```

Description:

Take acotangent of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = acot(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_DATASOURCE
```

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

acotd(obj) is the inverse cotangent, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acoth** ↑

```
obj = acoth(obj)
```

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acoth(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► **acsc** ↑

```
obj = acsc(obj)
```

Description:

Take inverse csc of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = acsc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acscd** ↑

```
obj = acscd(obj)
```

Description:

acscd(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acscd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE
```

► **acsch** ↑

```
obj = acsch(obj)
```

Description:

acsch(obj) is the inverse hyperbolic cosecant of the elements of obj

Input:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Output:

```
- obj : An object of class nb_ts, nb_cs or nb_data
```

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **addDataset** ↑

```
obj = addDataset(obj, dataset, NameOfDataset, startObs, variables)
```

Description:

Makes it possible to add one dataset to a existing nb_data object.

Caution : If more than one dataset is added to an object the link to the data source is broken!

Input:

```
- obj : An object of class nb_data
```

```
- dataset : One of the following:
```

```
> A string with the name of excel spreadsheet.  
(With or whitout extension)
```

```

> A string with the name of a .mat file
  (With or whitout extension)

> A numerical matrix.

- NameOfDataset : Must be a string with the dataset name. If not
  given (or given as '') the default name
  Database<jj> is given. Where jj is the number
  of added datasets of the object, including the
  dataset you add with this method.

When fetching from FAME this input could be a
vintage tag. I.e. you could specify the vintage
you want to fetch from FAME of the given
series. If the vintage tag does not exist it
will fetch the last vintage before the given
vintage tag.

- startObs      : Number indicatating the observation number. 1 is
  default

- variables     : A cell array of strings with the names of the
  variables you want to add. Only needed when a
  numerical matrix of data, an tseries object, an
  nb_math_ts object or when data from FAME is
  added to the object.

  Must have the same size as the data you add.
  (Second dimension)

When fetching data from FAME this input decides
which variables to fetch.

```

Output:

- obj : An nb_data object with the added dataset.

Examples:

```

obj = nb_data();
obj = obj.addDataset('excelFileName');
obj = obj.addDataset('excelFileName2','A dataset name');
obj = obj.addDataset('matFileName');

```

See also:

[nb_data](#), [addDatasets](#), [structure2Dataset](#)

Written by Kenneth S. Paulsen

► **addDatasets ↑**

```

obj = addDatasets(obj,datasets,NameOfDatasets,startObs, ...
  variables)

```

Description:

Makes it possible to add more datasets to a existing nb_data object.

Input:

Same input as of the constructor but only cell arrays are supported for the inputs 'datasets' and 'NameOfDatasets'. 'NameOfDatasets' could be given as a empty cell; {};

Output:

- obj : An nb_data object now with the added datasets.

Examples:

See also:

[nb_data](#), [addDataset](#), [structure2Dataset](#)

Written by Kenneth S. Paulsen

► **addPageCopies ↑**

obj = addPageCopies(obj,num)

Description:

Add copies of the data of an object and append it as new datasets of the object. Only possible if an object has only one page.

Input:

- obj : An object of class nb_data
- num : Number of copies to be appended

Output:

- obj : An object of class nb_data with the added copies of the first dataset in the object.

Examples:

```
data = nb_data([2,2;2,2],'',1,{'Var1','Var2'});
data = data.addPageCopies(10);
```

Written by Kenneth S. Paulsen

► **addPages** ↑

```
obj = addPages(obj,DB,varargin)
```

Description:

Add all pages from different nb_data objects.

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_data
- DB : An object of class nb_data
- varargin : Optional number of objects of class nb_data

Output:

- obj : The object itself with added datasets from the objects given by varargin.

Be aware: If the added datasets does not contain the same variables or observations, the data of the nb_data object with the tightest window will be expanded automatically to include all the same observations and variables. (the added data will be set as nan)

Examples:

```
obj = obj.addPages(DB);  
obj = obj.addPages(DB1,DB2,DB3);
```

Written by Kenneth S. Paulsen

► **addPostfix** ↑

```
obj = addPostfix(obj,postfix)  
obj = addPostfix(obj,postfix,vars)
```

Description:

Add a postfix to all the variables in the nb_ts, nb_cs or nb_data object, or a a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- postfix : The postfix to add to all the variables of the object. Must be a string
- vars : A cellstr of the variable to add the prefix to. Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with the postfix added to the variables selected

Examples:

```
obj = obj.addPostfix('_NEW');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **addPrefix ↑**

```
obj = addPrefix(obj,prefix)
obj = addPrefix(obj,prefix,vars)
```

Description:

Add a prefix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- prefix : The prefix to add to all the variables of the object. Must be a string
- vars : A cellstr of the variable to add the prefix to. Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with a prefix added to the selected variables

Example:

```
obj = obj.addPrefix('NEMO');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **addVariable** ↑

```
obj = addVariable(obj,startObsOfData,Data,Variable)
```

Description:

Add a new variable to a existing nb_data object (including all pages)

Input:

- obj : An object of class nb_data
- startObsOfData : The start observation of the data of the added variable. Must be an integer. Cannot be outside the window of the dates of the object. If empty, 1 is default.
- Data : The data of the added variable. Cannot be outside the window of the data of the object. Must be a double.
- Variable : A string with the added variable name. Cannot be the same as a existing variable. (Then use setValue instead)

Output:

- obj : An object of class nb_data with the added variable

Examples:

```
obj = addVariable(obj,'2012Q1',[2;2;2;2],'newVariable');
```

Written by Kenneth S. Paulsen

► **and** ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the and operator has evaluated all the data elements of the object.
The data will be a logical matrix

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **append ↑**

```
obj = append(obj,DB,method)
```

Description:

Append a nb_data object to another nb_data object.

Caution:

- It is possible to merge datasets with the same variables as long as they represent the same data or have different observation ids.
- If the datasets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they the one object can append the other.
- In contrast to the vertcat method, this method will keep the first inputs end obs and cut all the shared observations of the appended object.

Input:

- obj : An object of class nb_data
- DB : An object of class nb_data

Output:

- obj : An nb_data object where the datasets from the two nb_data objects are merged

Examples:

```
obj = append(obj,DB)
obj = obj.append(DB)
```

Written by Kenneth S. Paulsen

► **appendFromAnotherPage** ↑

```
obj = appendFromAnotherPage(obj,page)
```

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_dataSource object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_dataSource object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **asCell** ↑

```
cellMatrix = asCell(obj,strip)
```

Description:

Return the data of the object as a cell matrix.
(In the dyn_ts format, i.e. looks like a excel spreadsheet.)

Input:

- obj : An object of class nb_data
- strip :
 - 'on' : Strip all observation where all the variables has no value.
 - 'off' : Does not strip all observation where all the variables has no value. Default.

Output:

- `cellMatrix` : The `nb_data` objects data transformed to a cell.

Examples:

```
cellMatrix = obj.asCell();
cellMatrix = obj.asCell('on');
```

Written by Kenneth S. Paulsen

► **asCellForMPRReport ↑**

```
cellMatrix = asCellForMPRReport(obj, roundoff)
```

Description:

Return the data of the object as a cell matrix, which is rounded off to two decimals and striped

Input:

- `obj` : An object of class `nb_data`
- `roundoff` : 1 if you want to round off to 2 decimals, 0 otherwise. 1 is default.

Output:

- `cellMatrix` : The objects data transformed to a cell.

Examples:

```
cellMatrix = obj.asCellForMPRReport();
cellMatrix = obj.asCellForMPRReport(0);
```

See also:

[asCell](#)

Written by Kenneth S. Paulsen

► **asec ↑**

```
obj = asec(obj)
```

Description:

Take inverse sec of the data stored in the `nb_ts`, `nb_cs` or `nb_data` object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asecd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sin of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees,
of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **assignNan** ↑

```
obj = expand(obj,variables,value)
```

Description:

Assign all the nan observation to the value input.

Input:

- obj : An object of class nb_dataSource.
- variables : A char or a cellstr with the variables to assign nan values.
- value : A scalar number.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj = nb_ts.rand('2012Q1',10,2,3);
obj(1:2,1,:) = nan;
obj = assignNan(obj,'Var1',0)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the inverse tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees, of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **autocorr** ↑

```
obj = autocorr(obj,numLags)
obj = autocorr(obj,numLags,errorBound,alpha)
```

Description:

Compute the sample autocorrelation function (ACF) of all the variables stored in the nb_data object, and return a nb_data object with the results.

Reference:

[1] Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. Time Series Analysis: Forecasting and Control. 3rd edition. Upper Saddle River, NJ: Prentice-Hall, 1994.

Input:

- obj : An object of class nb_data.
Caution : This method does not handle nan or infinite for any of the stored data of the nb_data object!
- numLags : Positive integer indicating the number of lags of the ACF to compute. If empty or missing, the default is to compute the ACF at lags 1. Since ACF is symmetric about lag zero, negative lags are ignored. Must be a scalar.
- errorBound : A string. The data must be stationary. Either:
 - > 'asymptotic' : Using asymptotically derived formulas.
 - > 'bootstrap' : Calculated by estimating the data generating process. And use the fitted model to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
 - > 'blockbootstrap' : Calculated by blocking the observed series and using these blocks to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
- alpha : Significance value. As a scalar. Default is 0.05.

Optional inputs:

- 'maxAR' : As an integer. See the nb_arima function. Default is 3.
- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default

```
        is 'aicc'.
```

- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- obj : An nb_data object with the autocorrelations stored in the dataset 'Autocorrelations'.
If the errorbounds are to be calculated they are stored in the datasets 'Error bound (lower)' and 'Error bound (upper)'

Examples:

```
obj = autocorr(obj)
obj = autocorr(obj,10)
```

See also:

[nb_data](#), [nb_ts](#), [nb_cs](#), [corr](#), [nb_autocorr](#)

Written by Kenneth Sæterhagen Paulsen

► **bkfilter** ↑

```
obj = bkfilter(obj,low,high)
```

Description:

Do band pass filtering of all the dataseries of the object.
(Returns the gap). Will strip nan values when calculating the filter

Input:

- obj : An object of class nb_data
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_data with the band pass filtered data.

Examples:

```
obj = bkfilter(obj,8,32);
obj = obj.bkfilter(8,32);
```

See also:

[bkfilter1s](#)

Written by Kenneth S. Paulsen

► **bkfilter1s** ↑

```
obj = bkfilter1s(obj,low,high)
```

Description:

Do one sided band pass-filtering of all the dataseries of the object. (Returns the gap) Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_data
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_data with the one-sided band pass filtered data.

Examples:

```
obj = obj.bkfilter1s(12,24);
obj = bkfilter1s(obj,12,24);
```

See also:

[bkfilter](#)

Written by Kenneth S. Paulsen

► **breakLink** ↑

```
obj = breakLink(obj)
```

Description:

Break link to source

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callfun** ↑

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s). The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb.DataSource.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nb.DataSource.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb.DataSource objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource object. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d      = nb_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)
```

See also:

[nb_data.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callop** ↑

```
obj = callop(obj,another,func)
obj = callop(obj,another,func,type)
```

Description:

Call mathematical operator on the data of the object. In contrast to calling the operator itself on the object, this method does not try to match the same variables from the two inputs, but instead operate in the same way as mathematical operators do on double matrices (bsxfun). The time domain is matched though!

Caution: The following operators are supported:

```
> @plus or 'plus'
> @minus or 'minus'
> @times or 'times'
> @rdivide or 'rdivide'
> @power or 'power'
```

Input:

- obj : An object of class nbDataSource.
- another : An object of class nbDataSource.
- func : One of the above listed functions.
- type : Either:
 - > 'keep' : Keep the variable names of the object with most variables. If they have the same number of variables, they are taken from the first object.
 - > 'rename' : The new variable names represent the operation that has been done to the separate series. So if you divide an object with one series named 'Var1' with another object with one series named 'Var2', the new name will be 'Var1./Var2'. (Default)

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb.DataSource method. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,1),'''2011Q1',{'Var2'});
d3 = callop(d1,d2,@minus);

d1 = nb_ts(rand(10,1),'''2012Q1',{'Var1'});
d2 = nb_ts(rand(10,2),'''2011Q1',{'Var2','Var3'});
d3 = callop(d1,d2,@minus);
```

See also:

[nb_data.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► callstat ↑

```
obj = callstat(obj,func)
obj = callstat(obj,func,varargin)
```

Description:

Call a in-built or user defined function on the data of the nb.DataSource object(s) that goes from the data of the object have more than one variable to only having one. E.g. when taking the mean over a set of variables.

Input:

- obj : An object of class nb.DataSource.

- func : A function handle (or one line char) that maps a nObs x nVar x nPage double to nObs x 1 x nPage double, or that maps a nObs x nVar x nPage double to nObs x nVar x 1 double.

Optional input:

- 'name' : Set the name of the new variable, as a one line char. Default is to use the str2func on the provided function handle.

- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- obj : An object of class nb_dataSource with only one variable.

Examples:

```
d = nb_ts.rand('2012Q1',10,3);
d1 = callstat(d,@(x)mean(x,2),'name','mean')
```

See also:

[nb_data.mean](#), [nb_data.std](#), [nb_data.var](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► ceil ↑

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = ceil(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► conj ↑

```
obj = conj(obj)
```

Description:

conj(obj) is the complex conjugate of the elements of obj.
For a complex element x of obj, conj(x) = REAL(x) - i*IMAG(x).

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = conj(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► corr ↑

```
obj = corr(obj,varargin)
```

Description:

Calculate the correlation of the series of the nb_data object

Caution: Calculates the correlation matrix of the variables of the nb_data object, if no optional inputs are given. If 'lags' is given this function calculates the correlation with the number of wanted lags for each variable with itself. (And only with itself)

Input:

- obj : An object of class nb_data

Optional input ('propertyName',propertyValue):

- 'lags' : An integer with the number of lags you want to calculate the correlation for. (Here for each variable)

Output:

- obj : An nb_cs object with the wanted correlation

Examples:

```
corrMatrix = corr(obj);
corrMatrix = corr(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► **cos** ↑

```
obj = cos(obj)
```

Description:

Take cos of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

cosd(obj) is the cosine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cosd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cosh** ↑

```
obj = csch(obj)
```

Description:

`cosh(obj)` is the hyperbolic cosine of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cot** ↑

```
obj = cot(obj)
```

Description:

Take cotangent of the data stored in the `nb_ts`, `nb_cs` or `nb_data` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = cot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

`cotd(obj)` is the cotangent of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = cotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **coth** ↑

```
obj = coth(obj)
```

Description:

`coth(obj)` is the hyperbolic cotangent of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = coth(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cov** ↑

```
obj = cov(obj,varargin)
```

Description:

Calculate the covariances of the timeseries of the nb_data object

Caution: Calculates the covariance matrix of the variables of the nb_data object, if no optional inputs are given. If 'lags' is given this function calculates the covariance with the number of wanted lags for each variable with itself. (And only with itself)

Input:

- obj : An object of class nb_data

Optional input ('propertyName',propertyValue):

- 'lags' : An integer with the number of lags you want to calculate the covariance for. (Here for each variable)

Output:

- obj : An nb_cs object with the wanted covariances

Examples:

```
covMatrix = cov(obj);
covMatrix = cov(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► **createObsDummy** ↑

```
obj = createObsDummy(obj,nameOfDummy,obs,condition)
```

Description:

Creates and adds a dummy variable to the nb_data object. Will test the conditions given the date provided.

Input:

- obj : An object of class nb_data.

- nameOfDummy : Name of the added dummy variable.

- obs : The obs to test. As an integer.
- condition : The conditions to test.
 - '<' : True before.
 - '>' : True after.
 - '<=' : True before and including.
 - '>=' : False before and including.
 - '==' : True for the given period.
 - '~=': True for all but the given period.

Output:

- obj : An object of class nb_data. (With the added dummy variable)

Examples:

```
obj = nb_ts([1,2;-3,1;-1,3],'',1,['Var1','Var2']);
obj = createObsDummy(obj,'Dummy',2,'>');
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **createVarDummy** ↑

```
obj = createVarDummy(obj,nameOfDummy,varargin)
```

Description:

Creates and adds a dummy variable to the nb_data object. Will test the conditions given for the variable provided by the testedVariable input.

Input:

- obj : An object of class nb_data.
- nameOfDummy : Name of the added dummy variable.
- varargin : The conditions to test.
 - '<' : Less than. Second input must be a double (scalar), an nb_data object

(if more variables it must be less than all the stored variables) or a string with the variable to test against. The third input must either be '!' or '&' indicating if you want to check that this condition and ('&') or or ('|') the next is satisfied. The and operator(s) will have presedens over the or operator.

- '>' : Greater than. (Or else same as '<')
- '<=' : Less than or equal to. (Or else same as '<')
- '>=' : Less than or equal to. (Or else same as '<')
- '==' : Equal to. (Or else same as '<')
- '~=': Not equal to. (Or else same as '<')

Output:

- obj : An object of class nb_data. (With the added dummy variable)

Examples:

```
obj = nb_data([1,2;-3,1;-1,3],'',1,['Var1','Var2']);  
obj = createVarDummy(obj,'Dummy','Var1','>',0,'&');  
obj = createVarDummy(obj,'Dummy','Var1','>','Var2','&');  
obj = createVarDummy(obj,'D2','Var1','>',0,'|','Var2','<',1,'&');  
obj = createVarDummy(obj,'D2','Var1','>',0,'&','Var2','<',1,'&');
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **createVariable ↑**

```
obj = createVariable(obj,nameOfNewVariable,expression)  
obj = createVariable(obj,nameOfNewVariable,expression,parameters,varargin)
```

Description:

Create variable(s) and add them to the nb_data object dataset(s)

Caution :

- This function only support methods of the class nb_math_ts for the expressions.

Input:

- obj : An object of class nb_data
- nameOfNewVariable : The created variable names as a cell array or a string.
Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the data of all the created variables.
As a cell array or a string.
Must have the same size as the 'nameOfNewVariables' input.
- parameters : A struct with the parameters that can be used in the expressions. Caution: They will be added as series, so you need to use elementwise operators (.*, ./, .^)!

Optional input:

- 'overwrite' : true or false. Set to true to allow for overwriting of variables already in the data of the object. Default is false.
- 'warning' : true or false. Set to true to give warning if expressions fail (instead of error). Will result in series having all nan value, when warning is thrown.

Output:

- obj : An nb_data object with the created variable(s) added.

Examples:

```
obj = obj.createVariable('var3','var1./var2');  
  
obj = obj.createVariable({'var3','var4',...},...  
{'var1./var2','var2./var1',...});
```

Written by Kenneth S. Paulsen

► csc ↑

```
obj = csc(obj)
```

Description:

Take csc of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cscd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csch** ↑

```
obj = csch(obj)
```

Description:

`csch(obj)` is the hyperbolic cosecant of the elements of `obj`

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cumnansum** ↑

`obj = cumnansum(obj)`

Description:

Cumulative sum of the series of the object. Ignoring nan values.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An `nb_ts`, `nb_cs` or `nb_data` object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumnansum(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **cumprod** ↑

`obj = cumprod(obj, dim)`

Description:

Cumulativ product of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : A nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative product of the old objects 'data' property.

Examples:

```
obj = cumprod(obj);  
obj = cumprod(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumsum(obj);  
obj = cumsum(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cutAtObs** ↑

```
obj = cutAtObs(obj, endObs)
```

Description:

Cut the series of the variables of the object.

Input:

- obj : An object of class nb_data
- endDates : A nPages x nVars double array.

Output:

- obj : An object of class nb_data

Written by Kenneth S. Paulsen

► **deleteMethodCalls** ↑

```
obj = deleteMethodCalls(obj, c)
```

Description:

Delete some method calls of the object. The object needs to be updatable.

Caution: To set method calls use setMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- sourceNr : The source index to delete the methods from
- methodNrs : A index with the methods to delete.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

See also:

[setMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteVariables** ↑

```
obj = deleteVariables(obj,deletedVar)
```

Description:

Delete variables from the current nb_ts, nb_data or nb_cs object (letter size has something to say)

If some of the variables you specify in deletedVar does not exist in the obj, this will not have anything to say for the obj. (The variables cannot be deleted, because the variables do not exist.)

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- deletedVar : Must be a char or a cellstr

Delete all the variable names, given in the char array or cellstr array deletedVar, of current the object.

NB! If deletedVar is just one string and include a * as the last letter, this function will delete all the variable names, from the object, that start with the letters which is in front of the * in the string.

i.e. obj = deleteVariables(obj,'DPQ*');

Deletes all the variables in the obj which has variable names starting with 'DPQ'.

If deletedVar is just one string and include a * as the first letter, this function will delete all the variable names, from the object, that include the letters which follows in the string.

i.e. obj = deleteVariables(obj, '*shift');

Deletes all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_ts, nb_data or nb_cs object with the variables specified in deletedVar deleted.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});
obj = deleteVariables(obj, {'Var1','Var2'});
obj = deleteVariables(obj, char('Var1','Var2'));
obj = deleteVariables(obj, obj.variables);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_dataSource object.
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_dataSource object.

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATASOURCE

► **deptcat** ↑

```
obj = deptcat(obj,varargin)
```

Description:

Depth concatenation (add pages of different objects) (No short notation)

Be aware: If the added datasets does not contain the same variables or obs, the data of the nb_data object with the tightest window will be expanded automatically to include all the same dates and variables. (the added data will be set as nan)

Input:

- obj : An object of class nb_data
- varargin : Optional number of nb_data objects

Output:

- obj : An object of class nb_ts where all the objects datasets are added into.

Examples:

```
obj = obj.deptcat(DB);  
obj = obj.deptcat(DB,DB2,DB3);
```

See also:

[addPages](#)

Written by Kenneth S. Paulsen

► **detrend** ↑

```
[obj,trendObj] = detrend(obj,method)
```

Description:

Detrending nb_data object.

Input:

- obj : As an nb_data object.
- method : Either {'linear'} or 'mean'.

Output:

- obj : As an nb_data object with the detrended data.

Examples:

```
obj = detrend(obj,'linear');
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **diff** ↑

```
obj = diff(obj)  
obj = diff(obj,lags)  
obj = diff(obj,lags,skipNaN)
```

Description:

Calculate diff, using the formula: $x(t)-x(t-lag)$ of all the series of the nb_data object.

Input:

- obj : An object of class nb_data
- lags : The number of lags in the diff formula, default is 1.
- skipNaN : - 1 : Skip nan while using the diff operator. (E.g. when dealing with working days.)
 - 0 : Do not skip nan values. Default.

Output:

- obj : An nb_data object with the calculated series stored.

Examples:

```
obj = diff(obj);  
obj = diff(obj,4);
```

See also:

[nb_data](#)

Written by Kenneth S. Paulsen

► disp ↑

`disp(obj)`

Description:

Display object (On the commandline)

Input:

- obj : An object of class nb_data

Output:

The dataset displayed on the command, as an excel spreadsheet

Examples:

```
obj (Note without semicolon)
```

Written by Kenneth S. Paulsen

► **double** ↑

```
dataOfObject = double(obj)
```

Description:

Get the data of the object as a double matrix

Caution : If the data of the object is represents as an object of class nb_distribution the mean is returned.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- dataOfObject : The data of the nb_ts, nb_cs or nb_data object

Examples:

```
dataOfObject = double(obj)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **ecdf** ↑

```
obj = ecdf(obj,varargin)
```

Description:

A method for creating an nb_data object with the Empirical (Kaplan-Meier) cumulative distribution function. It utilizes the inbuilt ecdf function to estimate the distribution. See ecdf for more information on the optional inputs for the function.

Input:

- obj : An nb_data object with the data.

Optional input:

- 'recursive' : True or false. If true, the CDF will be estimated recursively. Default is false.
- 'recStart' : Sets the start obs for the recursive estimation. Default is the start obs of the obj + 9 observations. As a scalar integer.
- 'recEnd' : Sets the end obs for the recursive estimation. Default is the end obs of the object. As a scalar integer.
- 'estStart' : Sets the start of the estimation sample. As a scalar integer.

Output:

- obj : A numIter x numVar nb_data object where each observation point contains a nb_distribution object with the estimated parameters of the distribution.

Examples:

```
f = nb_data(randn(100,1),'',1,['v1']);
obj = f.ecdf('recStart',80,'recEnd',100, 'recursive',1);
```

See also:

[nb_data.ksdensity](#)

Written by Kenneth Sæterhagen Paulsen

► **egrowth ↑**

```
obj = egrowth(obj,lag)
```

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$ of all the timeseries of the nb_data object.

Input:

- obj : An object of class nb_data
- lag : The number of lags in the growth formula, default is 1.

Output:

- obj : An nb_data object with the calculated series stored.

Examples:

```
obj = egrowth(obj);
obj = egrowth(obj,4);
```

See also:

[growth](#)

Written by Kenneth S. Paulsen

► **empty** ↑

```
obj = empty(obj)
```

Description:

Empty the existing nb_data object.

Input:

- obj : An object of class nb_data

Output:

- obj : An empty nb_data object

Examples:

```
obj = empty(obj)
```

Written by Kenneth S. Paulsen

► **epcn** ↑

```
obj = epcn(obj,lag)
```

Description:

Calculate exact percentage growth, using the formula:
100*(x(t) - x(t-1))/x(t-1) of all the timeseries of the nb_data
object.

Input:

- obj : An object of class nb_data

- lag : The number of lags in the growth formula, default is 1.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = epcn(obj); % period-on-period growth  
obj = epcn(obj,4); % 4-periods growth
```

Written by Kenneth S. Paulsen

► eq ↑

```
a = eq(a,b)
```

Description:

Test if the one object is equal (==) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► estimateDist ↑

```
dist = estimateDist(obj,type)
```

Description:

Estimate distributions of the series in the nb_data object.

Caution : This method strip all nan values.

Input:

- obj : An object of class nb_data.
- type : The type of distribution to estimate. See the 'type' property of the nb_distribution class for the supported types. (Some may not be possible to estimate using 'mle' and/or 'mme'.) Default is 'normal';
 - Caution : If estimator is equal to 'kernel', this input is not used.
 - Caution : If type is set to 'hist', no estimator is used but the returned nb_distribution object instead represent a histogram.
- estimator : Either 'mle' (maximum likelihood), 'mme' (methods of moments) or 'kernel' (normal kernel density estimation). Default is 'mle'.
- dim : The dimension to estimate densities over. Either 1, 2 or 3. Default is 1.
- output : Either 'nb_distribution' (default) or 'nbDataSource'. If 'nbDataSource' is chosen the output will be a nb_data object if dim equal 2 or 3, while it will be a nb_cs object if dim equal 1.

Optional input:

- varargin : Optional input given to the nb_ksdensity function. Please see help for that function.

Output:

- dist : An object of class nb_distribution, nb_data or nb_cs.

See also:

[nb_distribution](#)

Written by Kenneth Sæterhagen Paulsen

► [exp ↑](#)

`obj = exp(obj)`

Description:

Take exp of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **expand ↑**

```
obj = expand(obj,newStartDate,newEndDate,type,warning)
```

Description:

Expand the current nb_ts object with more dates and data

Input:

- obj : An object of class nb_data
- newStartDate : The wanted new start obs of the data.
- newEndDate : The wanted new end obs of the data.
- type : Type of the appended data :
 - 'nan' : Expanded data is all nan (default)
 - 'zeros' : Expanded data is all zeros
 - 'ones' : Expanded data is all ones
 - 'rand' : Expanded data is all random numbers
 - 'obs' : Expand the data with first observation (before) or last observation after
- warningOff : Give 'off' to suppress warning if no expansion has taken place. Both 'newStartObs' and 'newEndObs' is inside the window.

Output:

- obj : An nb_data object with expanded span

Examples:

```
obj = expand(obj,1,100);
obj = expand(obj,1,100,'zeros');
```

Written by Kenneth S. Paulsen

► **expandPeriods** ↑

```
obj = expandPeriods(obj,periods,type)
```

Description:

Expand the data of the object by the number of wanted periods.

Input:

- obj : An object of class nb_data.
- periods : The number of periods to expand by. If negative the added periods will be added before the first observation.
- type : Type of the appended data :
 - 'nan' : Expanded data is all nan (default)
 - 'zeros' : Expanded data is all zeros
 - 'ones' : Expanded data is all ones
 - 'rand' : Expanded data is all random numbers
 - 'obs' : Expand the data with first observation (before) or last observation after

Output:

- obj : An object of class nb_data.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

► **expml** ↑

```
obj = expml(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, `e` to the `obj`.
`expm1(obj)` computes `exp(obj)-1`, compensating for the roundoff in
`exp(obj)`.
(For small real `X`, `expm1(X)` should be approximately `X`, whereas the
computed value of `EXP(X)-1` can be zero or have high relative error.)

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data` where the
data are equal to `e.^obj.data-1`

Examples:

```
obj = expm1(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **extMethod** ↑

```
obj = extMethod(obj,method,variables,postfix,varargin)
```

Description:

This method can call a `nb.DataSource` method on a selected variables, and
merge the result with the old object by adding a postfix to the new
variables.

This method is the same as calling:

```
obj1 = window(obj,'','variables');
method = str2func(method);
obj = method(obj,varargin{:});
obj1 = addPostfix(obj1,method);
obj = merge(obj,obj1);
```

But without replicating the `links` property unnecessary many times!

Caution : If the object is not updateable this will result in the same as
the code above!

Input:

- `obj` : An object of class `nb_ts`, `nb_data` or `nb_cs`
- `method` : A str with the method to call. Must be a method of the
`nb_ts`, `nb_data` or `nb_cs` class that does not change other
dimensions then the second dimension!

- variables : A cellstr with the variables of the object to call the method on. Must be included in the object.
- postfix : A string with the postfix. If empty, the old variables will be replaced by the new ones.

Optional inputs:

These will be the inputs casted to the method except the object itself.

Extra for nb_ts and nb_data:

- '<start>' : The start date/obs of the function evaluation. Given as the second input to the window method call.
- '<end>' : The end date/obs of the function evaluation. Given as the third input to the window method call.

Output:

- obj : A nb_ts, nb_data or nb_cs object

Examples:

```
obj = nb_ts.rand(1,10,3);
obj = extMethod(obj,'lag',{'Var1'},'lag1',1)
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **floor** ↑

```
obj = floor(obj)
```

Description:

floor(obj) rounds the data elements of obj to the nearest integers towards minus infinity

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = floor(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE
```

► **fromRise_ts** ↑

```
data = nb_data.fromRise_ts(obj)
```

Description:

Transform from an RISE ts object or a struct of RISE ts objects to a nb_data object.

Input:

- obj : An object of class ts, or a structure of ts objects.

Output:

- data : An nb_data object. Be aware that the time dimension is stripped when using this transformation.

Examples:

```
data = nb_data.fromRise_ts();
```

Written by Kenneth S. Paulsen

► **ge** ↑

```
a = ge(a,b)
```

Description:

Test if the one object is greater than or equal to (\geq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a >= b;
obj = 2 >= b;
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get the properties of the object

Input:

- obj : An object of class nb_data
- propertyName : Name of the property you want to get:
 - 'data' : The data of the object as a double
 - 'dataNames' : Name of the object datasets, as cellstr array
 - 'endObs' : The end obs of the object as double
 - 'numberOfDatasets' : Number of datasets (pages) of the object. Size of the third dimension
 - 'numberOfObservations' : Number of observations of the object. Size of the first dimension
 - 'numberOfVariables' : Number of variables of the object. Size of the second dimension
 - 'startObs' : The start obs of the object as double
 - 'variables' : The variables of the object, given as a cellstr array
 - 'links' : A structure of the data source links.
 - 'sorted' : true or false
 - 'updateable' : 1 if updateable, 0 otherwise.

Output:

```
propertyValue : The property value of the object for the wanted  
                  property
```

Examples:

```
dataAsDouble = get(obj,'data');  
variables    = get(obj,'variables')
```

Written by Kenneth S. Paulsen

► getClassOfData ↑

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_dataSource.

Output:

- cl : Name of the class that the data of the object is stored as.

See also:

[nb_data.isDistribution](#), [nb_data.isDouble](#)
[nb_data.isBoolean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getCode ↑

```
code = getCode(obj,preamble)  
code = getCode(obj,preamble,'inputName',inputValue,...)
```

Description:

Get the LaTeX code of a table representing the data of the object

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_data
- preamble : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'combine' : If the nb_data object consist of two or three pages (datasets) this option can be used to combine all the data in one table. Either 1 or 0. Default is 0.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:

'Huge', 'huge', 'LARGE', 'Large', 'large',
'normalsize' (default), 'small',
'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.

- 'justification' : Sets the justification of the caption of the table. Must be a string.

Either 'justified','centering' (default),
'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.

- language : Either 'english' (default) or 'norwegian' ('norsk').

- lookUpMatrix :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';  
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
lookUpMatrix = {  
  
 'mnemonics1','englishDescription1','norwegianDescription1';  
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined
char, which will result in multi-line text of the
table.

- 'orientation' : The orientation of the table. Either:
 - > 'horizontal' : The dates as columns
 - > 'vertical' : The types as columns
- 'precision' : Sets the precision of the numbers in the
table. Must be a string. Default is '%3.2f'.
I.e. two decimals.
- 'rotation' : The rotation of the table in LaTeX. Either:
 - > 'sidewaystable' : A sideways table
 - > 'landscape' : Rotate the whole page
 - > '' : No rotation
- 'rowColor1' : Color of every second row of the table
starting from the second row. Same options
as was the case for the 'firstRowColor'
option.
- 'rowColor2' : Color of every second row of the table
starting from the third row. Same options
as was the case for the 'firstRowColor'
option.
- 'rowColor3' : Color of every third row of the table
starting from the fourth row. Only an option
in combination with the 'combine' input set
to 1 and that the nb_ts object has 3 pages.
Same options as was the case for the
'firstRowColor' option.

Output:

- code : A char with the tex code of the table

Examples:

```
obj = nb_data(rand(10,3),'',1,['Var1','Var2','Var3']);  
code = obj.getCode(1)
```

See also:

[nb_data](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **getCreatedVariables ↑**

```
created = getCreatedVariables(obj)
```

Description:

Get a N x 2 table of the variables created using the createVariable method. The first column list the created variables, while the second column list the expressions.

Input:

- obj : An object that is a subclass of nb_dataSource.

Output:

- created : A N x 2 cell array. See description of this method for more.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getMethodCalls ↑

```
[s,c] = getMethodCalls(obj)
```

Description:

Get all method calls as a cell matrix. The object needs to be updatable to get a list of methods called on the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.
- c : A cell matrix with a table of the method called on the object and its input. Will return {} if the object is not updatable.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► getRealEndObs ↑

```
realEndObs = getRealEndObs(obj)
realEndObs = getRealEndObs(obj,type)
```

Description:

Get the real end obs of the nb_data object. I.e the last observation which is not nan or infinite.

Input:

- obj : An object of class nb_data
- type : Either 'any' (default) or 'all'.

Output:

- realEndData : The last observation of the object which is not nan or infinite. As a double

Examples:

```
realEndObs = obj.getRealEndObs();  
realEndObs = obj.getRealEndObs('all');
```

Written by Kenneth S. Paulsen

► **getRealStartObs ↑**

```
realStartObs = getRealStartObs(obj)  
realStartObs = getRealStartObs(obj,format,type)
```

Description:

Get the real start obs of the nb_data object. I.e the first observation which is not nan or infinite.

Input:

- obj : An object of class nb_data
- type : Either 'any' (default) or 'all'.

Output:

- realStartObs : The first observation of the object which is not nan or infinite. As a double

Examples:

```
realStartObs = obj.getRealStartObs();  
realStartObs = obj.getRealStartObs('all');
```

Written by Kenneth S. Paulsen

► **getSource** ↑

```
sourceType = getSource(obj)
sourceType = getSource(obj,type)
```

Description:

Get source type.

Input:

- obj : An object of class nb_dataSource.
- type : Give 'program' to get the general source. Default is to give the specific type of source.

Output:

- sourceType : A one line char with the source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSourceList** ↑

```
s = getSourceList(obj)
```

Description:

Get the source list of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getVariable** ↑

```
variableData = getVariable(obj,variableName,startObs,endObs, pages)
```

Description:

Get the data of a variable, as a double

Input:

- obj : An object of class nb_data
- variableName : The variable you want the data of. Must be given as a string (name of variable) or a cellstr.
- startObs : Start obs of the return data of the given variable. As an integer
- endObs : End obs of the return data of the given variable. As an integer
- pages : Pages of the return data of the given variable. As a double or logical array, e.e. [1:2] or [1,3,5].

Output:

- variableData : The data of the variable you have given. Return [] if not found.

Examples:

```
variableData = getVariable(obj,'Var1');  
variableData = getVariable(obj,'Var1',1);  
variableData = getVariable(obj,'Var1',1,2);  
variableData = getVariable(obj,'Var1',1,2,[1:3]);
```

Written by Kenneth S. Paulsen

► **growth ↑**

```
obj = growth(obj,lag)
```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$ of all the series of the nb_data object.

Input:

- obj : An object of class nb_data
- lag : The number of lags in the approx. growth formula, default is 1.

Output:

- obj : An nb_data object with the calculated series stored.

Examples:

```
obj = growth(obj);
obj = growth(obj,4);
```

See also:

[egrowth](#)

Written by Kenneth S. Paulsen

► [gt](#) ↑

```
a = gt(a,b)
```

Description:

Test if the one object is greater then (>) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nbDataSource or a scalar number.
- b : An object of class nbDataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nbDataSource object where the data element are logical.

Examples:

```
obj = a > b;
obj = 2 > b;
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hasVariable** ↑

```
[ret,location] = hasVariable(obj,variable)
```

Description:

Test if an nb_CS object has a given variable

Input:

- obj : An object of class nb_CS
- variable : The variable name as a string. Can also be a cellstr with more variables.

Output:

- ret : 1 if found otherwise 0
- location : The location of the variable if found. Otherwise [].

Examples:

```
ret = obj.hasVariable('Var1')
[ret,location] = hasVariable(obj,'Var1')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hdi** ↑

```
obj = hdi(obj,perc)
```

Description:

Calculate the highest density interval (hdi) over the third dimension of the data of the object.

Input:

- obj : An object of class nb_data
- limits : A double with the wanted mass of the hdi of the data. E.g. 0.9, or [0.10,0.20].

Output:

- obj : A nb_data object with the limits of the hdis stored as different pages of the object. See the dataNames property for the ordering.

Be aware that the hdis may not be overlapping!

Written by Kenneth Sætherhagen Paulsen

► **head** ↑

`head(obj,nRows,page)`

Description:

Display the first n rows of a nb_data object.

Input:

- obj : An nb_data object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_data](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **histcounts** ↑

`obj = histcounts(obj,interval,centered)`

Description:

Order the observation (dim1) into bins using the specified interval

Input:

- obj : An object of class nb_data
- interval : A 1 x nBins double
- centered : A 1 x nBins double with the centered bins value.

Output:

- obj : An object of class nb_data

Examples:

```
obj = nb_data.rand(1,100,4,2);
obj = histcounts(obj,0.1:0.1:1)
obj = histcounts(obj,0.1:0.1:1,0.05:0.1:1)
```

See also:

[nb_histcount](#)

Written by Kenneth Sæterhagen Paulsen

► **histogram** ↑

```
data          = histogram(obj)
[data,plotter] = histogram(obj,M)
```

Description:

Create a histogram of a object containing only one variable and one page!

Input:

- obj : A nb_dataSource object with size N x 1 x 1.
- M : The number of bins of the histogram.

Output:

- data : A nb_cs object with size M x 1 x 1.
- plotter : A nb_graph_cs object with the histogram plotted.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **horzcat** ↑

```
obj = horzcat(a,varargin)
```

Description:

Horizontal concatenation of nb_data objects. ([a,b]) This is only possible if the different objects contain differen variables or have variables with the same values (start and/or end obs can differ). Uses the merge method.

If the start and/or end obs differ, nan values will be appended.

Input:

- a : An object of class nb_data
- varargin : Optional number of nb_data objects

Output:

- obj : An nb_data object with all the variables from the different nb_data object merge into one dataset

Examples:

```
obj = [a,b];
obj = [a,b,c];
```

See also:

[merge](#)

Written by Kenneth S. Paulsen

► **hpfilter** ↑

```
obj = hpfilter(obj,lambda)
```

Description:

Do hp-filtering of all the dataseries of the nb_data object.
(Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_data
- lambda : The lambda of the hp-filter

Output:

- obj : An nb_data object with the hp-filtered series.

Examples:

```
gap = hpfilter(data,3000);
trend = data-gap;
```

See also:

[hpfilter](#)

► **hpfilter1s** ↑

```
obj = hpfilter1s(obj,lambda)
```

Description:

Do one-sided hp-filtering of all the dataseries of the nb_data object. (Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_data
- lambda : The lambda of the hp-filter

Output:

- obj : An nb_data object with the hp-filtered timeseries.

Examples:

```
gap = hpfilter1s(data,3000);
trend = data-gap;
```

See also:

[hpfilter](#)

Written by Kenneth S. Paulsen

► **iegrowth** ↑

```
obj = iegrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact growth, i.e. the inverse method of the egrowth method of the nb_data class

Input:

- obj : An object of class nb_data
 - InitialValues : A nb_data object with a bigger window than obj, or a scalar or a double with the initial values of the indices. Must be of the same size as the number of variables of the nb_data object. If not provided 100 is default.
- Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_data object with the indices.

Examples:

```
obj = iegrowth(obj);
obj = iegrowth(obj,100);
obj = iegrowth(obj,[78,89]);
```

See also
`nb_data.egrowth`, `nb_data.igrowth`, `nb_data.growth`

Written by Kenneth S. Paulsen

► **iepcn ↑**

```
obj = iepcn(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact percentage growth, i.e. the inverse method of the epcn method of the nb_data class

Input:

- obj : An object of class nb_data
- InitialValues : A nb_data object with a bigger window than obj, or a scalar or a double with the initial values of the indices. Must be of the same size as the number of variables of the nb_data object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input

must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_data object with the indicies.

Examples:

```
obj = iepcn(obj);
obj = iepcn(obj,100);
obj = iepcn(obj,[78,89]);
```

See also:

[nb_data.epcn](#)

Written by Kenneth S. Paulsen

► [igrowth ↑](#)

```
obj = igrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of log approx. growth, i.e. the inverse method of the growth method of the nb_data class

Input:

- obj : An object of class nb_data
- InitialValues : A nb_data object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_data object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_data object with the indicies.

Examples:

```
obj = igrowth(obj);
obj = igrowth(obj,100);
obj = igrowth(obj,[78,89]);
```

See also
nb_data.growth, nb_data.egrowth, nb_data.iegrowth

Written by Kenneth S. Paulsen

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_data object. Will discard leading and trailing nan values.

Input:

- data : An nb_ts, nb_cs or nb_data object.
- method :
 - 'nearest' : Nearest neighbor interpolation
 - 'linear' : Linear interpolation. Default
 - 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - 'pchip' : Shape-preserving piecewise cubic interpolation
 - 'cubic' : Same as 'pchip'
 - 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : An nb_ts, nb_cs or nb_data object with the interpolated data.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ipcn** ↑

```
obj = ipcn(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method of the nb_data class

Input:

- obj : An object of class nb_data
- InitialValues : A nb_data object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_data object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_data object with the indicies.

Examples:

```
obj = ipcн(obj);
obj = ipcн(obj,100);
obj = ipcн(obj,[78,89]);
```

See also:

[nb_data.pcn](#)

Written by Kenneth S. Paulsen

► **isBoolean ↑**

```
ret = isBoolean(obj)
```

Description:

Check if the data of the nb_dataSource object is of class logical (boolean, i.e. true or false).

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isCrossSectional** ↑

ret = isCrossSectional(obj)

Description:

Test if this object is a cross sectional data object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_cs, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isDistribution** ↑

ret = isDistribution(obj)

Description:

Check if the data of the nb_dataSource object is of class nb_distribution

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isDouble** ↑

```
ret = isDouble(obj)
```

Description:

Check if the data of the nb_dataSource object is of class double (numeric)

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isTimeseries** ↑

```
ret = isTimeseries(obj)
```

Description:

Test if a nb_dataSource object is a time series object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_ts, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isUpdateable** ↑

```
ret = isUpdateable(obj)
```

Description:

Test if this object is updateable.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : 1 if updateable, otherwise 0.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isempty** ↑

ret = isempty(obj)

Description:

Test if a nb_ts, nb_cs or nb_data object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

ret = isempty(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isfinite** ↑

obj = isfinite(obj)

Description:

Test if each element of a nb_ts, nb_cs or nb_data object is finite.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_ts, nb_cs or nb_data object is nan.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isscalar** ↑

```
ret = isscalar(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is a scalar. Will always return 0, as an object of class nb_ts, nb_cs or nb_data is not a scalar.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : false

Examples:

```
0 = isscalar(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepPages** ↑

```
obj = keepPages(obj, pages)
```

Description:

Keep pages of the current nb_ts object

Input:

- obj : An object of class nb_ts
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty a empty nb_ts object is returned.

Can also be a cellstr with the names to keep, or a logical array with length equal to the number of pages (datasets).

Output:

- obj : An nb_ts object with the pages specified in the input are kept.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **keepVariables** ↑

```
obj = keepVariables(obj, keptVar)
```

Description:

Keep variables from the current nb_data object (letter size has something to say)

If some of the variables you specify in keptVar does not exist in the object, this will not have anything to say for the object. (The variables cannot be kept, because the variables do not exist.)

Input:

- obj : An object of class nb_data
- keptVar : Must be a char or a cellstr

Keeps all the variable names, given in the char array or cellstr array keptVar, of current the nb_ts object.

NB! If keptVar is just one string and include a * as the last letter, this function will keep all the variable names, from the DB, that start with the letters which is in front of the * in the string.

i.e. obj = keepVariables(obj,'DPQ*');

Keep all the variables in the obj which has variable names starting with 'DPQ'.

If keptVar is just one string and include a * as the first letter, this function will keep all the variable names, from the DB, that include the letters which follows in the string.

i.e. obj = keepVariables(obj, '*shift');

Keeps all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : An nb_data object with the variables specified in input 'keptVar' kept, all the others are deleted.

Examples:

```
obj = keepVariables(obj, {'Var1','Var2'});  
obj = keepVariables(obj, char('Var1','Var2'));  
obj = keepVariables(obj, DB.variables);
```

Written by Kenneth S. Paulsen

► **ksdensity ↑**

```
obj = ksdensity(obj,varargin)
```

Description:

A method for creating an nb_data object with the estimated kernel density of a sample. It can estimate a single probability density function, as well as recursive estimation.

Input:

- obj : An nb_data object with the data.

Optional input:

- 'recursive' : True or false. If true, the CDF will be estimated recursively. Default is false.
- 'recStart' : Sets the start obs for the recursive estimation. Default is the start obs of the obj + 9 observations. As a scalar integer.
- 'recEnd' : Sets the end obs for the recursive estimation. Default is the end obs of the object. As a scalar integer.
- 'estStart' : Sets the start of the estimation sample. As a scalar integer.
- Rest of the optional input id given to the nb_ksdensity function.

Output:

- obj : A numIter x numVar nb_data object where each observation point contains a nb_distribution object with the estimated parameters of the distribution.

Examples:

```
f = nb_data(randn(100,1),'',1,['v1']);
obj = f.ksdensity('recStart',80,'recEnd',100, 'recursive',1);
```

See also:

[nb_data.ecdf](#)

Written by Kenneth Sæterhagen Paulsen

► **kurtosis** ↑

```
obj = kurtosis(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the kurtosis of each timeseries. The result will be an object of class nb_data where all the non-nan values of all the series are being set to their kurtosis.

The output can also be set to be a double or a nb_cs object consisting of the kurtosis values of the data.

Input:

- obj : An object of class nb_data
- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N
 - Where N is the sample length.
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their kurtosis.
 - > 'double' : Get the kurtosis values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the kurtosis values as nb_cs object. The kurtosis will be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the kurtosis over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_dataObj = kurtosis(obj);
double      = kurtosis(obj,0,'double');
nb_csObj   = kurtosis(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► lag ↑

```
obj = lag(obj,periods,varargin)
```

Description:

Lag the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_data
- periods : Lag the data with this number of periods. Default is 1 period.

Optional input:

- 'cut' : true (cut sample to fit old sample length) or false (append new observations). Default is true.

Output:

- obj : An nb_data object where the data lagged by number of periods given by the input periods.

Examples:

```
obj = lag(obj,1);
```

Written by Kenneth S. Paulsen

► **lastObservation ↑**

```
obj = lastObservation(obj,number)
```

Description:

Get the last (real) observation(s) of the object.

Input:

- obj : An object of class nb_data.
- number : Number of elements to return. Default is 1, i.e only the last. As a integer.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► **le** ↑

```
a = le(a,b)
```

Description:

Test if the one object is less than or equal to (\leq) the other object elementwise and return a nbDataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nbDataSource or a scalar number.
- b : An object of class nbDataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nbDataSource object where the data element are logical.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **lead** ↑

```
obj = lead(obj,periods,varargin)
```

Description:

Lead the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_data
- periods : Lead the data with this number of periods. Default is 1 period.

Optional input:

- 'cut' : true (cut sample to fit old sample length) or false (append new observations at the start). Default is true.

Output:

- obj : A nb_data object where the data leaded by number of periods given by the input periods.

Examples:

```
obj = lead(obj,1);
```

Written by Kenneth S. Paulsen

► **log ↑**

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on logs.

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log10 ↑**

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on (base 10) logs.

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► log1p ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► log2 ↑

```
obj = log2(obj)
```

Description:

`log2(obj)` is the base 2 logarithm of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► lt ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less than (`<`) the other object elemetswise and return a `nb_dataSource` object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- `a` : An object of class `nb_dataSource` or a scalar number.
- `b` : An object of class `nb_dataSource`, but must be of same subclass as `a`, or a scalar number.

Output:

- `a` : An `nb_dataSource` object where the data element are logical.

Examples:

```
obj = a < b;
obj = 2 < b;
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **map2Distribution** ↑

```
obj = map2Distribution(obj,dist)
obj = map2Distribution(obj,dist,varargin)
```

Description:

Map observation to the distributions given by dist.

The current distribution of the data is estimated using a kernel density estimator. The data must be strongly stationary. I.e. the unknown distribution from where the data has been drawn must not depend on time.

Input:

- obj : An object of class nb_ts.
- dist : An object of class nb_distribution with size 1 x
numberOfVariables.

Optional inputs:

- See the optional inputs of the nb_distribution.estimate function.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► **mavg** ↑

```
obj = mavg(obj,backward,forward,flag)
```

Description:

Taking moving average of all the series of the nb_data object

Input:

- obj : An object of class nb_data
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average
- flag : If set to true the periods that does not have enough observations forward or backward should be set to nan. Default is false.

Output:

- obj : An nb_data object storing the calculated moving average

Examples:

```
data = nb_data(rand(50,2)*3,'',1,['Var1','Var2']);  
  
mAvg10 = mavg(data,9,0); % (10 year moving average)
```

Written by Kenneth S. Paulsen

► max ↑

```
obj = max(obj,outputType,dimension)
```

Description:

Calculate the max of each timeseries. The result will be an object of class nb_data where all the non-nan values of all the timeseries are being set to their max.

The output can also be set to be a double or a nb_cs object consisting of the max values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their max.
 - > 'double' : Get the max values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the max values as nb_cs object. The max will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the max over.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_dataObj = max(obj);
double    = max(obj,'double');
nb_csObj = max(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **mean ↑**

```
obj = mean(obj,outputType,dimension,varargin)
```

Description:

Calculate the mean of each series. The result will be an object of class nb_data where all the non-nan values of all the timeseries are being set to their mean.

The output can also be set to be a double or a nb_cs object consisting of the mean values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their mean.
 - > 'double' : Get the mean values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mean values as nb_cs object. The mean will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the mean over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_tsObj = mean(obj);
double    = mean(obj,'double');
nb_csObj = mean(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **meanDiff** ↑

```
obj = meanDiff(obj)
```

Description:

Construct level variables that has the mean change for each period.

Input:

- obj : An object of class nb_data

Output:

- obj : A nb_data object where the data has been updated.

Written by Kenneth S. Paulsen

► **meanGrowth** ↑

```
obj = meanGrowth(obj)
```

Description:

Construct level variables that has the mean growth for each period.

Input:

- obj : An object of class nb_data

Output:

- obj : A nb_data object where the data has been updated.

Written by Kenneth S. Paulsen

► **median** ↑

```
obj = median(obj,outputType,dimension,varargin)
```

Description:

Calculate the median of each series. The result will be an object of class nb_data where all the non-nan values of all the series are being set to their median.

The output can also be set to be a double or a nb_cs object consisting of the median values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their median.
 - > 'double' : Get the median values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the median values as nb_cs object. The median will be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the median over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_dataObj = median(obj);
double      = median(obj,'double');
nb_csObj   = median(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **merge ↑**

```
obj = merge(obj,DB)
```

Description:

Merge two nb_data objects

Caution:

- It is possible to merge datasets with the same variables as long as they represent the same data or have different spans.
- If the datasets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they can merge.

Input:

- obj : An object of class nb_data
- DB : An object of class nb_data

Output:

- obj : An nb_data object where the datasets from the two nb_data objects are merged

Examples:

```
obj = merge(obj,DB)
obj = obj.merge(DB)
```

See also:

[horzcat](#), [vertcat](#)

Written by Kenneth S. Paulsen

► **merge2Series** ↑

```
obj = merge2Series(obj,var1,var2,new,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_dataSource.
- var1 : A one line char with the name of the base series.
- var2 : A one line char with the name of the appended series.
- new : Name of the new variable. If given as empty there will not be created a new series, but instead the var1 variable will be set to the new merged series. Default is empty.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_dataSource.

Examples:

```

obj           = nb_ts.rand('2012Q1',10,2,3);
obj(end-1:end,1,:) = nan;

obj1 = merge2Series(obj,'Var1','Var2','Var3')
obj2 = merge2Series(obj,'Var1','Var2')
obj3 = merge2Series(obj,'Var1','Var2','','diff','nLags',1)
obj4 = merge2Series(obj,'Var1','Var2','','diff','nLags',4)
obj5 = merge2Series(obj,'Var1','Var2','','level','date','2014Q1')

```

See also:

[nb_ts.merge](#), [nb_data.merge](#), [nb_cs.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► [min ↑](#)

obj = min(obj,outputType,dimension)

Description:

Calculate the min of each series. The result will be an object of class nb_data where all the non-nan values of all the timeseries are being set to their min.

The output can also be set to be a double or a nb_cs object consisting of the min values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their min.
 - > 'double' : Get the min values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the min values as nb_cs object. The min will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the min over.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_dataObj = min(obj);
double      = min(obj,'double');
nb_csObj   = min(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► minus ↑

```
obj = minus(obj,DBOrNum)
```

Description:

Binary subtraction (-).

Caution: Will subtract the data of the corresponding variables from the two objects. All the variables not in common will result in series with nan values.

When one of the inputs are a scalar this function will subtract each element of the data by this number or each element of the data will be subtracted from the scalar.

Input:

- a : An object of class nb_data or a scalar
- b : An object of class nb_data or a scalar

Output:

- obj : An nb_data object where all the variables are from the first object are subtracted from the other.

Or

An nb_data object where the scalar are subtracted from all the elements of the input object

Or

An nb_data object where all the elements of the input object are subtracted from scalar.

Examples:

```
obj = obj - anotherObj;  
obj = obj - 2;  
obj = 2 - obj;
```

See also:

[nb_data.plus](#), [nb_data.callop](#), [nb_data.callfun](#)

Written by Kenneth S. Paulsen

► **mldivide** ↑

```
obj = mldivide(a,b)
```

Description:

Matrix left division (\). See examples.

Input:

- a : An object of class nb_data.
- b : A scalar or a string.

Output:

- obj : An nb_data object.

Examples:

```
obj = nb_data.rand(1,10,3)
obj = obj\1      % 1/obj
obj = obj\'1'    % '1'/obj
obj = obj\'Var1' % 'Var1'/obj
```

See also:

[nb_data.mrdivide](#), [nb_data.rdivide](#)

Written by Kenneth S. Paulsen

► **mode ↑**

```
obj = mode(obj,outputType,dimension,varargin)
```

Description:

Calculate the mode of each series. The result will be an object of class nb_data where all the non-nan values of all the series are being set to their mode.

The output can also be set to be a double or a nb_cs object consisting of the mode values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their mode. Default
 - > 'double' : Get the mode values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mode values as nb_cs object. The mode will when this option is used only be

calculated over the number of observations.
(I.e. dimension set to 1.)

- dimension : The dimension to calculate the mode over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from
this method

Examples:

```
nb_dataObj = mode(obj);
double      = mode(obj,'double');
nb_csObj   = mode(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **mpower** ↑

```
obj = mpower(a,b)
```

Description:

Matrix power (^). Defined when raised by a scalar, a string
which either represent a variable or observation.

Input:

- a : An object of class nb_data
- b : A scalar or a string.

Output:

- obj : An nb_data object.

Examples:

```
obj  = nb_data.rand(1,10,2)
obj1 = obj^'1'
obj2 = obj^'Var1'
obj3 = obj^1
```

See also:

`nb_data.power`

Written by Kenneth S. Paulsen

► **mrdivide** ↑

`obj = mrdivide(a,b)`

Description:

Matrix right division (/). See examples.

Input:

- `a` : An object of class `nb_data`, a scalar number or a string representing a variable name or the observation number.
- `b` : An object of class `nb_data`, a scalar number or a string representing a variable name or the observation number.

Output:

- `obj` : An `nb_data` object.

Examples:

```
obj  = nb_data.rand(1,10,3)
obj1 = obj/1
obj2 = obj/'1'
obj3 = obj/'Var1'
obj4 = 1/obj
obj5 = '1'/obj
obj6 = 'Var1'/obj
```

See also:

`nb_data.mldivide`, `nb_data.rdivide`

Written by Kenneth S. Paulsen

► **mstd** ↑

`obj = mstd(obj,backward,forward)`

Description:

Taking moving standard deviation of all the series of the `nb_data` class

Input:

- obj : An object of class nb_data
- backward : Number of periods backward in time to calculate the moving std
- forward : Number of periods forward in time to calculate the moving std

Output:

- obj : An nb_data object storing the calculated moving standard deviation

Examples:

```
data = nb_data(rand(50,2)*3,'',1,['Var1','Var2']);  
  
mstd10 = mstd(data,9,0); % (10 year moving std)
```

Written by Kenneth S. Paulsen

► mtimes ↑

```
obj = mtimes(obj,Num)
```

Description:

Matrix multiplication (*) Defined for multiplication with a constant, a string which either represent a variable or obs.

Input:

- obj : An object of class nb_data or scalar
- b : An object of class nb_data or a scalar or a string.

Output:

- obj : An nb_data object where the number (date or variable as a vector) is multiplied with all the elements of the input objects data

Examples:

```
obj = 2*obj;  
obj = obj*2;  
obj = obj*'1'  
obj = obj*'Var1'
```

See also:

[nb_data.times](#)

Written by Kenneth S. Paulsen

► **nan** ↑

```
obj = nb_data.nan()
obj = nb_data.nan(start,obs,vars,pages,sorted)
```

Description:

Create an nb_data object with all data set to nan.

Input:

- start : The start obs of the created nb_data object.
- obs : The number of observation of the created nb_data object.
- vars : Either a cellstr with the variables of the nb_data object, or a scalar with the number of variables of the nb_data object.
- pages : Number of pages of the nb_data object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_data object.

Examples:

```
obj = nb_data.nan(1,10,{'Var1','Var2'});
obj = nb_data.nan(1,2,2);
obj = nb_data.nan(1,2,2,10);
```

See also:

[nb_data](#)

Written by Kenneth Sæterhagen Paulsen

► **nan2var** ↑

```
obj = nan2var(obj,testedVariables,assignVariable)
```

Input:

- obj : An object of class nb_data
- testedVariables : A cellstr with the variables you are setting equal to the assignVariable for the nan values found.
- assignVariable : A string with the name of the variable you are assigning the found nan values.

Output:

- obj : An nb_data object where all the nan values of the variables in the testedVariables input is set to the matching value of the variable given by the assignVariable input.

Examples:

```
obj = nb_data([1,2;nan,1;nan,3],'',1,{'Var1','Var2'});  
nan2var(obj,'Var1','Var2')
```

```
ans =
```

'Time'	'Var1'	'Var2'
'2012'	[1]	[2]
'2013'	[1]	[1]
'2014'	[3]	[3]

```
Written by Kenneth S. Paulsen
```

► ne ↑

```
a = ne(a,b)
```

Description:

Test if the one object is not equal to ($\sim=$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **not ↑**

```
a = not(a)
```

Description:

The not operator (~) .

Input:

- a : An object of class nb_dataSource.

Output:

- a : An object of class nb_dataSource. Where the not operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = ~a;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **objSize ↑**

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : - 1 : Number of observations
- 2 : Number of variables
- 3 : Number of datasets (pages)

Output:

- varargout : See the examples below

Examples:

```
dim = objSize(obj)
```

Where dim is 1 x 2 double where the first element is the number of observations/types and the second element is the number of variables

```
[dim1, dim2] = objSize(obj)
```

```
[dim1, dim2, dim3] = objSize(obj)
```

```
dim1 = objSize(obj,1)
dim2 = objSize(obj,2)
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **observations** ↑

```
obs = observations(obj,type)
```

Description:

Get all the observations of the nb_data object

Input:

- obj : An object of class nb_data
- type : A string. Either 'double' (default) or 'cellstr'.

Output:

- obs : Either a double or cellstr.

See also:

[nb_data](#)

► **ones** ↑

```
obj = nb_data.ones()  
obj = nb_data.ones(start,obs,vars,sorted)
```

Description:

Create an nb_data object with all data set to 1.

Input:

- start : The start obs of the created nb_data object. As an integer.
- obs : The number of observation of the created nb_data object.
- vars : Either a cellstr with the variables of the nb_data object, or a scalar with the number of variables of the nb_data object.
- pages : Number of pages of the nb_data object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_data object.

Examples:

```
obj = nb_data.ones(1,10,['Var1','Var2']);  
obj = nb_data.ones(1,2,2);  
obj = nb_data.ones(1,2,2,10);
```

See also:

[nb_data](#)

► **or** ↑

```
a = or(a,b)
```

Description:

The and operator (|).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the or operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

a = a | b;

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **packing ↑**

obj = packing(obj,packages,varargin)

Description:

Package variables by applying the wanted function. Default is summin (sum).

Input:

- obj : An object of class nb_ts.
- packages : A 2 x N cell matrix with groups of variables and the names of the groups. If you have not listed a variable it will be grouped in a group called 'Rest' (as long as 'includeRest' is not set to false).

Must be given in the following format:

```
{'group_name_1','...','group_name_N';
 name_1      ,...,name_N}
```

Where name_x must be a string with a name of the variable or a cellstr array with the variable names. E.g 'Var1' or {'Var1','Var2'}.

Optional inputs:

- 'includeRest' : Give false to not include the 'Rest' variable, which will represent all the variable that has not been packed.
- 'func' : Either a function_handle or a one line char with the name of the function to use. The first input to this function is a nObs x nVar double, the second is the dimension (which is always 2). Examples; 'sum', 'prod', @sum or @prod.

Output:

- obj : A object of class nb_ts.

Examples:

```
data      = nb_ts.rand('2012Q1',10,4);
packages = {'group1','group2';
            {'Var1','Var2'},{'Var3'}};

dataP1 = packing(data,packages)
dataP2 = packing(data,packages,'includeRest',false)
dataP3 = packing(data,packages,'func',@prod)
```

See also:

[nb_ts.createVariable](#), [nb_data.createVariable](#), [nb_cs.createVariable](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **parcorr ↑**

```
obj = parcorr(obj,numLags)
obj = parcorr(obj,numLags,errorBound,alpha)
```

Description:

Compute the sample partial autocorrelation function (PACF) of all the variables stored in the nb_data object, and return a nb_data object with the results.

Reference:

[1] Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. Time Series Analysis: Forecasting and Control. 3rd edition. Upper Saddle River, NJ: Prentice-Hall, 1994.

Input:

- obj : An object of class nb_data.

```

Caution : This method does not handle nan or
           infinite for any of the stored data
           of the nb_data object!

- numLags      : Positive integer indicating the number of lags
                  of the PACF to compute. If empty or missing, the
                  default is to compute the PACF at lags 1.
                  Since PACF is symmetric about lag zero, negative
                  lags are ignored. Must be a scalar.

- errorBound   : A string. The data must be stationary. Either:

    > ''          : No errorbounds calculated.

    > 'asymtotic' : Using asymptotically derived
                     formulas.

    > 'bootstrap'  : Calculated by estimating the
                     data generating process. And
                     use the fitted model to
                     simulate artificial
                     time-series. Then the error
                     bands are constructed by
                     the wanted percentile.

    > 'blockbootstrap' : Calculated by blocking the
                        observed series and using
                        these blocks to simulate
                        artificial time-series. Then
                        the error bands are
                        constructed by the wanted
                        percentile.

- alpha        : Significance value. As a scalar. Default is 0.05.

```

Optional inputs:

- 'maxAR' : As an integer. See the nb_arima function. Default is 3.
- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- obj : An nb_data object with the partial autocorrelations
 stored in the dataset 'Autocorrelations'.

 If the errorbounds are to be calculated they are stored
 in the datasets 'Error bound (lower)' and 'Error bound
 (upper)'

Examples:

```
obj = parcorr(obj)
obj = parcorr(obj,10)
```

See also:

[nb_data](#), [nb_cs](#), [corr](#), [nb_parcorr](#)

Written by Kenneth Sæterhagen Paulsen

► **pca** ↑

```
F = pca(obj)
[F,LAMBDA,R,varF,expl,c,sigma,e,Z] = pca(obj,r,method,varargin)
```

Description:

Principal Component Analysis of the data of the object.

Input:

- obj : An object of class nb_data.
- r : The number of principal component. If empty this number will be found by the Bai and Ng (2002) test, see the optional option crit below, i.e choose the CT part of the selection criterion; $\log(V(ii,F)) + CT$. Where;
 $V(ii,F) = \sum(\text{diag}(e_{ii}' * e_{ii}/T))/ii$
and e_{ii} is the residual when ii factors are used
- method : Either 'svd' or 'eig'. Default is 'svd'.
 - > 'svd' : Uses single value deomposition to construct the principal components. This is the numerically best way, as the 'eig' method could be very unprecise!
 - > 'eig' : Uses the eigenvalue decomposition approach instead

Optional inputs:

- 'crit' : You can choose between the follwing selection criterion
 - > 1: $\log(NT/NT1)*ii*NT1/NT;$
 - > 2: $(NT1/NT)*\log(\min([N;T]))*ii;$
 - > 3: $ii*\log(GCT)/GCT;$
 - > 4: $2*ii/T;$

```

> 5: log(T)*ii/T;
> 6: 2*ii*NT1/NT;
> 7: log(NT)*ii*NT1/NT; (default)

where NT1 = N + T, GCT = min(N,T), NT = N*T and ii = 1:rMax

- 'rMax' : The maximal number of factors to test for. Must be less than or equal to N. Default is N.

- 'trans' : Give 'demean' if you want to demean the data in the matrix X. Default. Give 'standardize' if you want to standardise the data in the matrix X. Give 'none' to drop any kind of transformation of the data in the matrix X.

Caution: To get back X when 'demean' is given you need to add the constant terms in c. I.e;

X = c(ones(T,1),1) + F*LAMBDA + e, e ~ N(0,R) (1*)

To get back X when 'standardise' you need to add the constant term and multiply with sigma. I.e.

X = c(ones(T,1),:) + F*LAMBDA.*sigma(ones(T,1),:) +
eps, eps ~ N(0,R*) (1*)

Be aware that the eps differ from e, as e is the residual from the standardised regression.

'output' : A string with the wanted output. Either 'F', 'LAMBDA', 'R', 'varF', 'expl', 'c', 'sigma', 'e'. See below for the matching output.

Caution: If provided it will be the only output from this function. I.e. return as F.

```

Output:

- F : The principal component, as a nb_data object.
- LAMBDA : The estimated loadings, as a nb_cs object
- R : The covariance matrix of the residual in (1), as a nb_cs object.
- varF : As a nb_cs object. Each column will be the variance of the corresponding column of F.
- expl : The percentage of the total variance explained by each principal component. As a 1 x r nb_cs object
- c : Constant in the factor equation. nb_cs object with size 1 x nvar.
- sigma : See 'trans'. nb_cs object with size 1 x nvar.
- e : Residual of the equation (1) above. As a nb_data object.

- Z : Normalized (and rebalanced) version of X.

See also:

[nb_pca](#)

Written by Kenneth Sæterhagen Paulsen

► **pcn** ↑

obj = pcn(obj,lag)

Description:

Calculate log approximated percentage growth. Using the formula
 $(\log(t+lag) - \log(t)) * 100$

Input:

- obj : An object of class nb_data
- lag : The number of lags. Default is 1.

Output:

- obj : An nb_data object with the log approximated percentage growth data stored.

Examples:

```
obj = pcn(obj); % period-on-period log approx. growth
obj = pcn(obj,4); % 4-periods log approx. growth
```

See also:

[growth](#), [egrowth](#), [ipcn](#)

Written by Kenneth S. Paulsen

► **percentiles** ↑

obj = percentiles(obj,perc)

Description:

Take the percentile over the third dimension of the data of the object.

Input:

- obj : An object of class nb_data
- perc : A double with the wanted percentiles of the data. E.g. 0.5 (median), or [0.05,0.15,0.25,0.35,0.5,0.65,0.75,0.85,0.95].

Output:

- obj : A nb_data object with the percentiles stored as different pages of the object. See the dataNames property for the ordering.

Written by Kenneth Sæterhagen Paulsen

► permute ↑

```
obj = permute(obj,variables)
```

Description:

Switch the second and third dimension of the nb_ts, nb_cs or nb_data object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- variables : A cellstr with size 1 x numberOfDatasets. Default is to use dataNames property. I.e. when not provided or if empty.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► plot ↑

```
plot(obj,varargin)
plotter = plot(obj,varargin)
```

Description:

Plot the data of the nb_dataSource object.

Caution: If the data of the object is of class nb_distribution this method will redirect to plotDist.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

- plotType : A string with either:

> 'graph' (Default)

> 'graphSubPlots'

> 'graphInfoStruct'

The name of the graphing method of the class nb_graph_ts, nb_graph_cs or nb_graph_data.

Caution: This is an optional input. If not given it is assumed that the optional input pairs are starting from the second input to this function.

- varargin : Same as the input to the method set() of the nb_graph_ts, nb_graph_cs or nb_graph_data class.

I.e. ..., 'inputName', inputValue, ...

Caution: If the data of the object is of class nb_distribution, see nb_ts.plotDist for the optional input pairs supported.

Output:

Plotted output

Examples:

plot(obj)

plot(obj, 'graphSubPlots')

plot(obj, 'startGraph', '2008Q1', 'endGraph', '2011Q2')

same as

plot(obj, 'graph', 'startGraph', '2008Q1', 'endGraph', '2011Q2')

See also:

[nb_graph_ts](#), [nb_ts.plotDist](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► [plus](#) ↑

obj = plus(obj, DBOrNum)

Description:

Binary addition (+).

Caution: Will add the data of the corresponding variables from the two objects. All the variables not in common will result in series with nan values.

When one of the inputs are a scalar this function will add each element of the data by this number.

Input:

- a : An object of class nb_data or a scalar
- b : An object of class nb_data or a scalar

Output:

- obj : An nb_ts object where all the variables from the two objects are added.

Or

An nb_data object where the scalar are added to all elements of the given nb_data object

Examples:

```
obj = obj + anotherObj;  
obj = obj + 2;  
obj = 2 + obj;
```

See also:

[nb_data.minus](#)

Written by Kenneth S. Paulsen

► pow2 ↑

obj = pow2(obj)

Description:

pow2(obj) raises the number 2 to the power of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► power ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^).

If both inputs are nb_ts objects this method will raise the data of the first object with the data from the second object. This will only be done for the variables which are found to be in both nb_ts objects. The rest of the variables will get nan values.

Input:

- obj : An object of class nb_data or a scalar
- DBOrNum : An object of class nb_data or a scalar

Output:

- obj : An nb_data object where the data from th one object are raised with the data from the other object

or

An nb_data object where all the elements are raised by the scalar DBOrNum.

or

An nb_data object where the scalar obj are raised by all the elements of the object DBOrNum.

Examples:

```
obj = a.^b;
obj = a.^2;
obj = 2.^a;
```

See also:

[nb_data.mpower](#), [nb_data.callop](#), [nb_data.callfun](#)

Written by Kenneth S. Paulsen

► **rand** ↑

```
obj = nb_data.rand(start,obs,vars,pages,dist,sorted)
```

Description:

Create an nb_data object with all data set to random number.

Input:

- start : The start obs of the created nb_data object. As an integer
- obs : The number of observation of the created nb_data object.
- vars : Either a cellstr with the variables of the nb_data object, or a scalar with the number of variables of the nb_data object.
- pages : Number of pages of the nb_data object.
- dist : A nb_distribution object to draw from.
- sorted : true or false. Default is true.

Output:

- obj : An nb_data object.

Examples:

```
obj = nb_data.rand('1',10,['Var1','Var2']);  
obj = nb_data.rand('2012Q1',2,2);  
obj = nb_data.rand('2012Q1',2,2,10,nb_distribution('type', 'normal'));
```

See also:

[nb_data](#)

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

```
obj = rdivide(obj,DB)
```

Description:

Right element-wise division (./)

Do right element-wise division of the matching variables of the two nb_data objects. If some variables are not found to be in both objects, the resulting object will have series with nan values only for these variables.

Input:

- a : An object of class nb_data
- b : An object of class nb_data

Output:

- obj : An object of class nb_data, where the data are the result of the element-wise division.

Examples:

```
obj = obj./DB;
```

See also:

[nb_data.mldivide](#), [nb_data.mrdivide](#), [nb_data.callop](#), [nb_data.callfun](#)

Written by Kenneth S. Paulsen

► **reIndex** ↑

```
obj = reIndex(obj,date,baseValue)
```

Description:

Reindex the data of the object to a new obs.

Input:

- obj : An object of class nb_data
- obs : The obs at which the data should be reindexed to.
(Reindexed to baseValue, default is 100)
- baseValue : A number to re-index the series to. Default is 100.

Output:

- obj : An object of class nb_data where all the objects series are reindexed to 100 at the given obs

Examples:

```
obj = reIndex(obj,5)
```

Written by Kenneth S. Paulsen

► **real** ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part of the elements in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **reallog** ↑

```
obj = reallog(obj)
```

Description:

Take the real natural logarithm of the data stored in the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = reallog(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **realpow** ↑

```
obj = realpow(obj,DBOrNum)
```

Description:

Same as nb_data.power.

See also:

[nb_data.power](#)

Written by Kenneth S. Paulsen

► **realsqrt** ↑

```
obj = realsqrt(obj)
```

Description:

realsqrt(obj) is the square root of the elements of obj.
An error is produced if obj contains negative elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = realsqrt(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **rename** ↑

```
obj = rename(obj,type,oldName,newName)
```

Description:

Makes it possible to rename variables and datasets. For objects of class nb_cs types is also possible to rename.

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variable', but not when the input is 'dataset' or 'types'.

Input:

- obj : An object of class nb_dataSource
- type : Either:
 - 'variable' : To rename a variable(s)
 - 'dataset' : To rename a dataset(s) (The only input that is supported for nb_cell)
 - 'types' : To rename a type(s) (only nb_cs)
- oldName : Case 1;
 - > The old variable/dataset/type name, as a string
- Case 2:
 - > The string which should be replaced in all the variable names of the database. And reorder things afterwards. This input must end with a *.
- newName : Case 1;
 - > The new variable/dataset/type name, as a string
- Case 2;
 - > The string which should replace the old string part given by the input 'oldName'.

Output:

- obj : An nb_dataSource object with the renamed variable(s), type(s) or dataset(s).

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.rename('variable','Var1','Var3');  
obj = obj.rename('variable','Var*','N_Var');  
obj = rename(obj,'variable',{'Var1','Var2'},{'VAR1','VAR2'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **renameMore** ↑

```
obj = renameMore(obj,type,oldNames,newNames)
```

Description:

Makes it possible to rename more variables, datasets and types (only nb_cs).

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variables', but not when the input is 'datasets'.

Caution : This is the same as looping over nb_dataSource.rename, but more efficient.

Caution : * syntax is not supported as is the case for nb_dataSource.rename

Input:

- obj : An object of class nb_dataSource.
- type : Either:
 - 'variables' : To rename a variables
 - 'datasets' : To rename a datasets (The only input that is supported for nb_cell)
 - 'types' : To rename a type (Only nb_cs)
- oldNames : The old variable/dataset/type names, as a cellstr.
- newName : The new variable/dataset/type names, as a cellstr.

Output:

- obj : An nb_dataSource object where the variable(s)/dataname(s)/type(s) of the object given as input is/are renamed.

Examples:

```
obj = nb_cs([22,24;27,28],'Dataset1',{'Exp','Imp'},{'Nor','Swe'})  
obj = renameMore(obj,'dataset',{'Dataset1'},{'tradeBal'});  
obj = renameMore(obj,'variable',{'Nor','Swe'},{'Norway','Sweden'});  
obj = renameMore(obj,'type',{'Exp','Imp'},{'Export','Import'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **reorder ↑**

```
obj = reorder(obj,newOrder,type)
```

Description:

Re-order the variables/types/datasets of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- newOrder : A cellstr with the new order of the variables/types/datasets.
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ret ↑**

```
obj = ret(obj)
obj = ret(obj,nlag)
obj = ret(obj,nlag,skipNaN)
```

Description:

Calculate return, using the formula: $x(t)/x(t-lag)$ of all the series of the nb_data object.

Input:

- obj : An object of class nb_data
- nlag : The number of lags in the return formula, default is 1.
- skipNaN : -1 : Skip nan while using the ret operator. (E.g. when dealing with working days.)
 - 0 : Do not skip nan values. Default.

Output:

- obj : An nb_data object with the calculated series stored.

Examples:

```
obj = ret(obj);
obj = ret(obj,4);
```

See also:

[nb_data](#), [nb_ts](#)

► **rlag** ↑

```
obj = rlag(obj,t,endObs)
```

Description:

Append data with use of the rolling lag operator

Input:

- obj : An object of class nb_ts
- t : The number of lags to use for the rolling window
- endObs : The new end observation

Output:

- obj : An object of class ts

Examples:

```
obj = nb_data(rand(20,1),'',1,'Var1')
obj = rlag(obj,4,30)
```

► **round** ↑

```
obj = round(obj,value,startObs,endObs,variables,pages)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_data object
- value : The rounding value. As a double. Default is 1.
- startObs : The start obs of the rounding. As an integer. Can be empty. Default is the start obs of the data.
- endObs : The end obs of the rounding. As an integer. Can be empty.

Default is the end obs of the data.

- variables : The variables to round. Can be empty. Default is to round all variables.
- pages : A numerical index of the pages you want to keep.

Output:

- obj : A nb_data object with the rounded numbers.

Written by Kenneth SÃ¶terhagen Paulsen

► **saveDataBase ↑**

saveDataBase(obj,varargin)

Description:

Save data of the object to (a) file(s)

If no optional input is given, i.e. obj.saveDataBase(). The data of the object is save down to xlsx file(s). Each dataset of the object is saved down to a excel spreadsheet with the name of the dataset (Taken from the 'dataNames' property)

Input:

- obj : An object of class nb_data
- Optional input (..., 'propertyName', PropertyValue, ...):
- 'saveName' : A string with the wanted name of the saved file.
 - 'ext' : > 'xlsx' : Saves the object down to a excel spreadsheet. Default
 - > 'matold' : Saves the data down to a mat file. The data is transformed into a structure, with the variables as the fieldnames, and the data as its fields. If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method with input old set to 1.
Must be combined with the optional input 'saveName'.
 - > 'mat' : Saves the data down to a mat file. The data is transformed into a structure, with the variables stored in the field 'variables', while the data of the variables will be saved as its fields with generic

names ('Var1','Var2' etc). If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method.

Must be combined with the optional input 'saveName'.

```

> 'mats'    : The object is saved down as a struct
               using the nb_data.struct function.

> 'txt'     : Saves the data down to a txt file.

- 'append'   : > 1 : Saves all the datasets (pages of the data)
               to one excel spreadsheet, but in separate
               worksheets. (Where the names of the
               worksheets are given by the dataNames
               property of the object.)
```

Works only when 'ext' input is set to
'xlsx'. (The default)

```

> 0 : The default saving method.

- 'strip'    : > 'on'  : Strip all observation dates where all
               the variables has no value. (is nan)

> 'off' : Default
```

Caution : Only an option when saved to excel.

Output:

The nb_data object saved to the wanted file format.

Examples:

Save the data to excel with the name 'test':
saveDataBase(obj,'saveName','test');

Save the data to excel with another date format:
saveDataBase(obj,'saveName','test','dateformat','xls');

Save the data to excel spreadsheet with the datasets as different worksheets:
saveDataBase(obj,'saveName','test','append',1);

Save the data to a .mat with the name 'test':
obj.savDataBase('ext','mat','saveName','test')

See also:

[asCell](#), [toStructure](#), [nb_readMat](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

► **sec ↑**

```
obj = sec(obj)
```

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **secd ↑**

```
obj = secd(obj)
```

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **set2nan** ↑

```
obj = set2nan(obj,testedVariable,settledVariable)
```

Input:

- obj : An object of class nb_data
- testedVariable : A string with the name of the variable you are testing for nan values
- settledVariable : A string with the name of the variable you are setting equal to nan where the testedVariable are in fact nan.

Output:

- obj : An nb_data object where the settledVariable are set equal to nan at all the same dates as the testedVariable is nan.

Examples:

```
obj = nb_data([1,2;nan,1;2,3],'',1,{'Var1','Var2'});  
  
obj = set2nan(obj,'Var1','Var2');  
  
obj =  
  
'Time'    'Var1'    'Var2'  
'2012'    [ 1]    [ 2]  
'2013'    [ NaN]    [ NaN]  
'2014'    [ 2]    [ 3]
```

Written by Kenneth S. Paulsen

► **setInfinite2NaN** ↑

```
obj = setInfinite2NaN(obj)
```

Description:

Set all elements that are isfinite to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setMethodCalls** ↑

```
obj = setMethodCalls(obj,c)
```

Description:

Set all method calls of the object with a cell matrix. The object needs to be updatable.

Caution: To delete method calls use deleteMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

- c : A cell matrix. See the output of getMethodCalls

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

See also:

[nb_data.deleteMethodCalls](#), [nb_data.getMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setValue ↑**

```
obj = setValue(obj,VarName,Value,startObsOfValues,...  
              endObsOfValues,pages)
```

Description:

Set some values of the dataset(s). (Only one variable at a time.) You cannot use obs outside the objects observations (i.e. outside obj.startObs:obj.endObs)

Input:

- obj : An object of class nb_data
- VarName : The variable name of the variable you want to set some values of. As a string
- Value : The values you want to assign to the variable. Must be a numerical vector (with the same number of pages as given by pages or as the number of dataset the object consists of.) Cannot be outside the window of the data of the object.
- startObsOfValues : A obs a an integer with the start obs of the data. If not given or given as a empty double it will assume that the setted data start at the startObs of the object.
- endObsOfValues : A obs a an integer with the end date of the data. If not given or given as a empty double it will assume that the setted data end at the endObs of the object.
- pages : At which pages you want to set the values of a variable. Must be a numerical index of the pages you want to set. E.g. if you want to set the values of the 3 first datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages must be set.

Output:

- obj : An nb_data object where the data of the given variable has been set.

Examples:

```
obj = nb_data([2;3;4],'',1,{'Var1'});  
  
obj = obj.setValue('Var1',[1;2;3]);  
obj = obj.setValue('Var1',2,2,2,1);  
obj = obj.setValue('Var1',[1;2;3],2,3);
```

See also
addVariable

Written by Kenneth S. Paulsen

► **setZero2NaN** ↑

```
obj = setZero2NaN(obj)
```

Description:

Set all elements that are 0 to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **sin** ↑

```
obj = sin(obj)
```

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = sin(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **sind** ↑

obj = sind(obj)

Description:

sind(obj) is the sine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = sind(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sinh** ↑

obj = sinh(obj)

Description:

sinh(obj) is the hyperbolic sine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **skewness ↑**

```
obj = skewness(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the skewness of each series. The result will be an object of class nb_data where all the non-nan values of all the series are being set to their skewness.

The output can also be set to be a double or a nb_cs object consisting of the skewness values of the data.

Input:

- obj : An object of class nb_data

- flag : Same as for the function skewness created by MATLAB.

- outputType :

> 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their skewness.

> 'double' : Get the skewness values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.

> 'nb_cs' : Get the skewness values as nb_cs object. The skewness will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)

- dimension : The dimension to calculate the skewness over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method
- dimension : The dimension to calcualate the skewness over.

Examples:

```
nb_dataObj = skewness(obj);
double      = skewness(obj,0,'double');
nb_csObj   = skewness(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► **sort ↑**

```
obj = sort(obj,mode,dim,variable)
```

Description:

Sort the wanted dimension (observations) of the data in the order given by the order input.

Caution : Sorting does not reorder the dataNames, variables or observations!

Input:

- obj : An object of class nb_data
- mode : 'descend' (default)
'ascend'
- dim : The dimension to sort. Either 1, 2 or 3. Default is 3.
- variable : Give a string with the variable to sort. The rest of the variables will then be reordered accordingly.

If empty (default), all variables are sorted.

Caution: Only an option for dim == 1

Output:

- obj : An object of class nb_ts

Examples:

```
in  = nb_ts(rand(10,2,10),'',1,['Var1','Var2']);
out = sort(in);
out = sort(in,'descend')
```

Written by Kenneth S. Paulsen

► **sortProperty** ↑

```
obj = sortProperty(obj,type)
```

Description:

Sort the variables/types/datasets of the object alphabetically.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **splitSeries** ↑

```
obj = splitSeries(obj,variable,obs,postfix,overlapping)
```

Description:

Split a series into two variables at a given observation.

Input:

- obj : An object of class nb_data
- variable : The variable(s) to split. Either a string or a cellstr.
- obs : The observation to split the series. As an integer.
- postfix : A string with the postfix of the new variable(s).
- overlapping : Indicate if you want the splitted series to be overlapping or not. As a logical. Default is true.

Output:

- obj : An object of class nb_data with the splitted series added.

See also:

[nb_ts](#), [nb_ts.splitSample](#)

Written by Kenneth Sæterhagen Paulsen

► **sqrt** ↑

`obj = sqrt(obj)`

Description:

`sqrt(obj)` is the square root of the elements of `obj`. Complex results are produced for non-positive elements.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

`out = sqrt(in);`

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **squeeze** ↑

`obj = squeeze(obj)`

Description:

Squeezes the different datasets (pages) of the given `nb_data` object. It will append the variables in each dataset with the given data name

Input:

- `obj` : An `nb_data` object. If the object has only one dataset (page) this method has nothing to do.

Output:

- obj : An nb_data object, where all the pages are squeezed together. I.e. all data will be stored in one dataset, and all variable names will get the dataset name appended.

Example:

```
obj = nb_data(ones(1,1,2),{'b','c'},2,{'Var1'})  
  
obj =  
  
(:,:,1) =  
  
    'Time'      'Var1'  
    '2012'      [ 1]  
  
(:,:,2) =  
  
    'Time'      'Var1'  
    '2012'      [ 1]  
  
sObj = squeeze(obj)  
  
sObj =  
  
    'Time'      'Var1_b'      'Var1_c'  
    '2012'      [ 1]          [ 1]
```

Written by Kenneth Sæterhagen Paulsen

► std ↑

```
obj = std(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the std of each series. The result will be an object of class nb_data where all the non-nan values of all the series are being set to their std.

The output can also be set to be a double or a nb_cs object consisting of the std values of the data.

Input:

- obj : An object of class nb_data
- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N

Where N is the sample length.
- outputType :

```

> 'nb_data': The result will be an object of class nb_data
    where all the non-nan values of all the
    timeseries are being set to their std.

> 'double' : Get the std values as doubles, where
    each column of the double matches the
    variable location in the 'variables' property.
    If the object consists of more pages the double
    will also consist of more pages.

> 'nb_cs' : Get the std values as nb_cs object.
    The std will when this option is used only
    be calculated over the number of observations.
    (I.e. dimension set to 1.)

- dimension : The dimension to calculate the std over.

```

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from
 this method

Examples:

```

nb_dataObj = std(obj);
double      = std(obj,0,'double');
nb_csObj   = std(obj,0,'nb_cs');

```

Written by Kenneth S. Paulsen

► **stdise ↑**

```
obj = stdise(obj,flag)
```

Description:

Standardise data of the object by subtracting mean and dividing
by std deviation.

Input :

- obj : An object of class nb_data
- flag : - 0 : normalises by N-1 (Default)
 - 1 : normalises by N

Where N is the sample length.

Output:

- obj : An nb_data object with the standardised data

Examples:

```
obj = stdise(obj);
```

Written by Kenneth S. Paulsen

► **stopSorting** ↑

```
obj = stopSorting(obj)
```

Description:

Stop sorting of variables.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where sorting is turned off.

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **strip** ↑

```
obj = strip(obj,startObs,endObs,variables)
```

Description:

Sets all values of the given variables of a nb_data object, between two given observations, to nan.

Input:

- obj : A nb_data object

- startObs : The starting point of the strip, must be either an nb_date object or a valid input to the toDate method of the nb_date class

- endObs : The starting point of the strip, must be either an nb_date object or a valid input to the toDate method of the nb_date class
- variables : A cellstring with the variables to perform the method on. Can be empty, in which case the method will be performed on all variables in the object.

Output:

- obj : A nb_data object

Examples:

```
strippedObj = strip(obj,'2011Q1','2012Q3',{'Var1','Var2'})
strippedObj = strip(obj,'2011Q1','2012Q3')
```

Written by Kenneth S. Paulsen

► **stripButLast** ↑

```
obj = stripButLast(obj,numPeriods,variables)
```

Description:

Sets all values of the given variables of a nb_data object, except a given amount of observations at the end, to nan.

Input:

- obj : A nb_data object
- numPeriods : The amount of observations to be left as-is in the object. If the input is given to 2, the returned object will have all it's observations set to nan except the last two observations.
- variables : A cellstring with the variables to perform the method on. Can be empty, in which case the method will be performed on all variables in the object.

Output:

- obj : A nb_data object

Examples:

```
strippedObj = stripButLast(obj,5',{'Var1','Var2'})
strippedObj = stripButLast(obj,'5')
```

Written by Eyo Herstad

► **struct** ↑

```
s = struct(obj)
```

Description:

Object to struct.

Input:

- obj : An nb_data object.

Output:

- s : A structure representing an nb_data object.

Written by Kenneth Sætherhagen Paulsen

► **structure2Dataset** ↑

```
obj = structure2Dataset(obj,givenStruct,NameOfDataset,startObs)
```

Description:

When you give a struct to the constructor of this class all the fields af that struct will be added as a new dataset, but this method add all the fields of the input struct as a variable in one dataset

If you only want to add a dataset from a structure in this way (no other datasets) intitialize an empty object, and then use this method.

Caution : The structure must include a field 'startObs', which must be an integer with the start observation.

Inputs:

- obj : An nb_data object, could be empty.
- givenStruct : The structure which contains the data of the added variable. Where the fieldnames will be the variables name of the added dataset + plus one field 'startObs' with the start obs of the dataset.
- NameOfDataset : Name of the added dataset. If not provided a default name will be given
- startObs : The start obs of the added structure. Only needed if the provided structure has no field

```
'startObs'.
```

Outputs:

- obj : An nb_data object with the added structure as a dataset.

Examples:

```
s = struct();
s.startDate = 1;
s.var1 = [2;2;2];
data = nb_data();
data = data.structure2Dataset(s)
```

Written by Kenneth S. Paulsen

► **subsasgn** ↑

```
varargout = subsasgn(obj,s,b)
```

Description:

Makes it possible to do subscripted assignment of the data of an object

Input:

- obj : An object of class nb_data
- s : The same input as when you do subscripted assignment one a double matrix
- b : The assign data at the given index s.

Output:

- obj : The assign data property setted of the given object.

Examples:

```
obj = nb_data([2;3;4],'',1,['Var1']);
obj(1:3,1) = [1;2;3]
obj =
{'Time'      'Var1'
 '2012'      [    1]
 '2013'      [    2]
 '2014'      [    3]
```

Written by Kenneth S. Paulsen

► **subsref** ↑

```
varargout = subsref(obj,s)
```

Description:

Makes it possible to do subscripted reference of the data of an object. See the examples for more

Using the method window, just using indexing instead

Input:

- obj : An object of class nb_data
- s : A structure on have to index the object

Output:

- obj : The indexed object.

Examples:

```
obj = obj(1,2:4,1)
```

same as

```
obj = obj(1,2:4)
```

Be aware that that obj(1,:) and obj(1,2:end) will also work

```
obj = obj('2')
```

same as

```
obj.window('2','2')
```

```
obj = obj('2','4')
```

same as

```
obj.window('2','4')
```

```
obj = obj('2','4',{'Var1','Var2',...})
```

same as

```
obj.window('2','4',{'Var1','Var2',...})
```

```
obj = obj('2','4',{'Var1','Var2',...},1:2)
```

```

same as

obj.window('2','4',{'Var1','Var2',...},1:2)

obj = obj('Var1','Var2',...)

same as

obj.window('','','{'Var1','Var2',...},1:obj.numberOfDatasets)

```

Written by Kenneth S. Paulsen

► sum ↑

```
obj = sum(obj,dim)
```

Description:

Takes the sum of the object series

Caution : All sums ignore nan values. If not all values is nan.

Input:

- obj : An object of class nb_data
- dim : The dimension to sum over, returns:
 - > dim = 1: Either :
 - > An nb_data object with all the elements of each variable set to their sum over the time horizon.
 - > An nb_cs with one type representing the sum over the obs horizon). The type is named 'sum'.
 - > dim = 2: An nb_data object with one variable representing the sum (Take the sum over the varibales) (Default)
 - > dim = 3: An nb_data object with only on page, representing the sum over all pages.
- output : Either 'nb_data' (default) or 'nb_cs'. Only important when summing over the 1 dimension.

Output:

- obj : Either an nb_data object or an nb_cs object representing the sum.

Examples:

```
obj      = nb_data([2,1;3,2;4,3],'', '1', {'Var1','var2'})  
obj =  
  
'Time'    'Var1'    'var2'  
'1'       [ 2]       [ 1]  
'2'       [ 3]       [ 2]  
'3'       [ 4]       [ 3]  
  
s1 = sum(obj,1,'nb_data')  
  
s1 =  
  
'Time'    'Var1'    'var2'  
'1'       [ 9]       [ 6]  
'2'       [ 9]       [ 6]  
'3'       [ 9]       [ 6]  
  
s1_cs = sum(obj,1,'nb_cs')  
  
s1_cs =  
  
'Types'   'Var1'    'var2'  
'sum'     [ 9]       [ 6]  
  
s2 = sum(obj,2)  
  
s2 =  
  
'Time'    'sum'  
'1'       [ 3]  
'2'       [ 5]  
'3'       [ 7]  
  
s2 = sum(obj,3)  
  
s2 =  
  
'Time'    'Var1'    'var2'  
'1'       [ 2]       [ 1]  
'2'       [ 3]       [ 2]  
'3'       [ 4]       [ 3]
```

Written by Kenneth S. Paulsen

► **summary ↑**

```
obj = summary(obj)
```

Description:

Summary of the series of the nb_data object

Input:

- obj : An object of class nb_data

Output:

- obj : the summary statistics - min, max, std, var, skewness and kurtosis

Written by Kenneth S. Paulsen

► **tail** ↑

`tail(obj,nRows,page)`

Description:

Display the last n rows of a nb_data object.

Input:

- obj : An nb_data object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_data](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **tan** ↑

`obj = tan(obj)`

Description:

Take the tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = tan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **tand** ↑

```
obj = tand(obj)
```

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

tanh(obj) is the hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tanh(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► **times** ↑

```
obj = times(obj,DB)
```

Description:

Element-wise multiplication (.*)

Caution : Will only multiply the data of corresponding variables of the two objects. I.e. variable not found to be in both object will result in series with all values set to nan.

Input:

- a : An object of class nb_data
- b : An object of class nb_data

Output:

- obj : An nb_data object the data results from element-wise multiplication of the data of the input objects.

Examples:

```
obj = obj.*DB;
```

See also:

[nb_data.mtimes](#), [nb_data.callop](#), [nb_data.callfun](#)

Written by Kenneth S. Paulsen

► **toMFile** ↑

```
string = toMFile(obj)  
toMFile(obj,filename)  
string = toMFile(obj,filename,varName)
```

Description:

Write a .m file to generate the current object.

Input:

- obj : An nb_data object.
- filename : The filename to write the code to. If empty no file will be written.
- varName : Name of decleared variable by this code. Default is data.

Output:

- string : A cellstr with the .m code to generate the current object.

Written by Kenneth Sæterhagen Paulsen

► toStructure ↑

```
retStruct = toStructure(obj)
retStruct = toStructure(obj,old)
```

Description:

Transform an nb_data obj to a structure

Input:

- obj : An object of class nb_data
- old : Indicate if old mat file format is wanted. Default is 0 (false).

Output:

- retStruct : A structure with fieldsnames given by the object variables and field given by the variables data. (Can have more than one page). And a own field with the start obs of the data. (As an integer.)

Examples:

```
obj = nb_data(ones(8,1),'',1,{'Var1'});
s = obj.toStructure()
s =
    Var1: [8x1 double]
    variables: {'Var1'}
    startObs: 1
    class: 'nb_data'
    userData: ''
    localVariables: []
    sorted: 1
```

See also:

[nb_data.struct](#)

Written by Kenneth S. Paulsen

► **tonb_cell** ↑

`nb_cell_DB = tonb_cell(obj)`

Description:

Transform from a nb_data object to a nb_cell object

Input:

- `obj` : An object of class nb_data
- `strip` : - 'on' : Strip all observations where all the variables has no value.
 - 'off' : Does not strip all observations where all the variables has no value. Default.

Output:

- `nb_cell_DB` : An object of class nb_cell

Examples:

`nb_cell_DB = obj.tonb_cell();`

Written by Kenneth S. Paulsen

► **tonb_cs** ↑

`nb_cs_DB = tonb_cs(obj)`

Description:

Transform from a nb_data object to a nb_cs object

Input:

```
- obj : An object of class nb_data

- strip : - 'on' : Strip all observations where all
           the variables has no value.

           - 'off' : Does not strip all observations
           where all the variables has no value.
           Default.
```

Output:

```
- nb_cs_DB : An object of class nb_cs
```

Examples:

```
nb_cs_DB = obj.tonb_cs();
```

Written by Kenneth S. Paulsen

► **tonb_ts ↑**

```
nb_ts_DB = tonb_ts(obj,startDate)
```

Description:

Transform from a nb_data object to a nb_ts object

Input:

```
- obj : An object of class nb_data

- startDate : Sets the start date of the nb_ts object. If empty or
               not given it is default to use startObs as the start date.
```

Output:

```
- nb_ts_DB : An object of class nb_ts
```

Examples:

```
nb_ts_DB = obj.tonb_ts();
```

Written by Kenneth S. Paulsen

► **uminus ↑**

```
obj = uminus(obj)
```

Description:

Unary minus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : A nb_ts, nb_cs or nb_data object where all the data are the unary minus of the input objects data

Examples:

```
obj = -obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **undiff ↑**

```
obj = undiff(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series diff. Inverse of the diff method.

Input:

- obj : An object of class nb_data

- initialValues : A nb_data object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_data object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series has been taken diff over.

Output:

- Output : An nb_data object with the indicies.

Examples:

```
obj = undiff(obj);
obj = undiff(obj,100);
obj = undiff(obj,[78,89]);
```

See also
nb_data.diff

Written by Kenneth S. Paulsen

► **unstruct** ↑

```
obj = nb_ts.unstruct(s)
```

Description:

Reverse of the struct method.

Input:

- s : See the s output of the struct method.

Output:

- obj : An nb_data object.

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,warningOff,inGUI)
```

Description:

This method will try to update the data of the nb_ts object.

Caution : It is only possible to update an nb_dataSource object which has updateable links to a FAME database or a full directory (path) to an excel spreadsheet or .mat file or the SMART database.

Caution : If you want to create a link to a specific worksheet of a excel file you must provide the extension!

Caution : All method calls done on the object are preserved.
(There exist some exception; and, or etc.)

Input:

- obj : An object of class nb_dataSource which are updatable.
- warningOff : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.

Output:

- obj : An object of class nb_dataSource with the data updated. If it is not possible a warning will be given.

Examples:

```
obj = nb_ts(['N:\Matlab\Utvikling\MATLAB_LIB\NEMO\'...
             'Documentation\Examples\example_ts_quarterly']);
obj = obj.update
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **uplus ↑**

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **var ↑**

```
obj = var(obj,outputType,dimension,varargin)
```

Description:

Calculate the var of each series. The result will be an object of class nb_ts where all the non-nan values of all the series are being set to their var.

The output can also be set to be a double or a nb_cs object consisting of the var values of the data.

Input:

- obj : An object of class nb_data
- outputType :
 - > 'nb_data' : The result will be an object of class nb_data where all the non-nan values of all the series are being set to their var.
 - > 'double' : Get the var values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the var values as nb_cs object. The variance will be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the var over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_tsObj = var(obj);
double   = var(obj,'double');
nb_csObj = var(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **vertcat** ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_data
- b : An object of class nb_data
- varargin : Optional numbers of objects of class nb_data

Output:

- a : An nb_data object where the data from the different objects are appended to each other.

Examples:

```
obj = nb_data(ones(2,1),'',1,['Var1']);  
aObj = nb_data(ones(2,1),'',3,['Var1']);  
m = [obj;aObj]
```

```
m =
```

'Time'	'Var1'
'1'	[1]
'2'	[1]
'3'	[1]
'4'	[1]

Written by Kenneth S. Paulsen

► **window ↑**

```
obj = window(obj,startObsWin,endObsWin,variablesWin,pages)
```

Description:

Narrow down window of the data of the nb_data object

Input:

- obj : An object of class nb_data
- startObsWin : The new wanted start obs of the data of the object. Must be an integer.
- endObsWin : The new wanted end obs of the data of the object. Must be an integer.

- variablesWin : A cellstr array of the variables you want to keep of the given object. If empty all the variables is kept.
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty all pages is kept.
I.e. `obj = obj.window('','',{ },1:3)`

Output:

- obj : An nb_data object where the datasets of the object is narrowed down to the specified window.

Examples:

```
obj = nb_data(ones(10,2,2),'',1',{'Var1','Var2'});
obj = obj.window(2)
obj = obj.window('',5)
obj = obj.window('','','',{'Var1'})
obj = obj.window('','','',{ },1)
```

See also:

[subsref](#)

Written by Kenneth S. Paulsen

► **writeTex** ↑

```
writeTex(obj,filename)
writeTex(obj,filename,'inputName', inputValue,...)
```

Description:

Writes the object data to a LaTeX table saved to a .tex file.

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_data
- filename : The name of the saved .tex file. With or without the extension.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'combine' : If the nb_data object consist of two or three pages (datasets) this option can be used to combine all the data in one table. Either 1 or 0. Default is 0.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:
 - 'Huge', 'huge', 'LARGE', 'Large', 'large',
 'normalsize' (default), 'small',
 'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.
- 'justification' : Sets the justification of the caption of the table. Must be a string.
 - Either 'justified','centering' (default),
 'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.
- language : Either 'english' (default) or 'norwegian'
 ('norsk').
- lookUpMatrix :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
lookUpMatrix = {

 'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined char, which will result in multi-line text of the table.
- 'orientation' : The orientation of the table. Either:
 - > 'horizontal' : The obs as columns
 - > 'vertical' : The types as columns
- 'precision' : Sets the precision of the numbers in the

table. Must be a string. Default is '%3.2f'. I.e. two decimals.

- 'rotation' : The rotation of the table in LaTeX. Either:
 - > 'sidewaystable' : A sideways table
 - > 'landscape' : Rotate the whole page
 - > '' : No rotation
- 'rowColor1' : Color of every second row of the table starting from the second row. Same options as was the case for the 'firstRowColor' option.
- 'rowColor2' : Color of every second row of the table starting from the third row. Same options as was the case for the 'firstRowColor' option.
- 'rowColor3' : Color of every third row of the table starting from the fourth row. Only an option in combination with the 'combine' input set to 1 and that the nb_ts object has 3 pages. Same options as was the case for the 'firstRowColor' option.

Output:

A saved .tex file with the LaTeX table

Examples:

```
obj = nb_ts(rand(10,3),'',1,{'Var1','Var2','Var3'});
obj.writeTex('test')
```

See also:

[nb_data](#)

Written by Kenneth Sæterhagen Paulsen

► **zeros ↑**

```
obj = nb_data.zeros()
obj = nb_data.zeros(start,obs,vars,pages,sorted)
```

Description:

Create an nb_data object with all data set to 0.

Input:

- start : The start obs of the created nb_data object. As an integer.
- obs : The number of observation of the created nb_data object.
- vars : Either a cellstr with the variables of the nb_data object, or a scalar with the number of variables of the nb_data object.
- pages : Number of pages of the nb_data object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_data object.

Examples:

```
obj = nb_data.zeros(1,10,{'Var1','Var2'});
obj = nb_data.zeros(1,2,2);
obj = nb_data.zeros(1,2,2,10);
```

See also:

[nb_data](#)

Written by Kenneth Sæterhagen Paulsen

■ nb_math_ts

Go to: [Properties](#) | [Methods](#)

A class representing timeseries data.

The difference between this and the nb_ts class is that all reference to the variables are removed. This makes some operators work as normal matrices. (Matrix multiplication, matrix division and matrix power of two object of class nb_math_ts is for example undefined for this class.)

That the object of this class has no reference to the object makes it easier to do datatransformations. E.g.

```
var1 = nb_math_ts([2;2;2;2],'2012Q1');
var2 = nb_math_ts([2;2;2;2],'2012Q1');
var3 = var1./var2;
var3 =
    '2012Q1'      [1]
    '2012Q2'      [1]
    '2012Q3'      [1]
    '2012Q4'      [1]
```

Caution : nb_ts and this class has not the same methods.

Constructor:

```

obj = nb_math_ts(data,startDate)

Input:

- data      :
    > A double with the data.

    > An object of class nb_ts

    > An object of class tseries (Needs the IRIS package)

- startDate :
    The start date of the data. Only needed when you give
    numerical data as one element of the input datasets (also
    when numerical data is given through a struct).

    Must be a string on the date format given below or an
    object which is of a subclass of the nb_date class.

    Dating convention:
    > Yearly      : 'yyyy'.           E.g. '2011'
    > Semiannualy : 'yyyySs'          E.g. '2011S1'
    > Quarterly   : 'yyyyQq'.          E.g. '2011Q1'
    > Monthly     : 'yyyyMm(m)'.      E.g. '2011M1',
                                '2011M11'
    > Daily       : 'yyyyMm(m)Dd(d)' E.g. '2011M1D1',
                                '2011M11D11'

    Caution: Must not be provided when the data input is of
              class nb_ts or tseries.

Output:

- obj      : An object of class nb_math_ts.

Examples:

obj = nb_math_ts([2,2;2,2],'2012Q1');

See also:

nb\_ts

Written by Kenneth Sæterhagen Paulsen

Properties:

- data    • dim1    • dim2    • dim3    • endDate    • startDate



- data ↑

The data of the object. As a double or logical

- dim1 ↑

The size of the first dimension of the data property. As a double.

```

- **dim2** ↑

The size of the second dimension of the data property. As a double.

- **dim3** ↑

The size of the third dimension of the data property. As a double.

- **endDate** ↑

The end date of data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_month, nb_quarter, nb_semiAnnual or nb_year

- **startDate** ↑

The start date of the data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_month, nb_quarter, nb_semiAnnual or nb_year.

Methods:

- [abs](#)
- [acosh](#)
- [acoth](#)
- [acsch](#)
- [appendFromAnotherPage](#)
- [asech](#)
- [asinh](#)
- [atanh](#)
- [bkfilter1s](#)
- [ceil](#)
- [corr](#)
- [cosh](#)
- [coth](#)
- [csc](#)
- [cumprod](#)
- [demean](#)
- [deptcat](#)
- [disp](#)
- [empty](#)
- [exp](#)
- [extrapolate](#)
- [ge](#)
- [getRealEndDate](#)
- [gt](#)
- [horzcat](#)
- [hpfilter1s](#)
- [igrowth](#)
- [acos](#)
- [acot](#)
- [acsc](#)
- [and](#)
- [asec](#)
- [asin](#)
- [atan](#)
- [avgOAF](#)
- [breakAdj](#)
- [conj](#)
- [cos](#)
- [cot](#)
- [cov](#)
- [csed](#)
- [cumsum](#)
- [demeanMoving](#)
- [detrend](#)
- [double](#)
- [epcn](#)
- [expand](#)
- [fillNaN](#)
- [get](#)
- [getRealStartDate](#)
- [h2l](#)
- [horzcatfast](#)
- [iegrowth](#)
- [interpolate](#)
- [acosd](#)
- [acotd](#)
- [acsed](#)
- [append](#)
- [asecd](#)
- [asind](#)
- [atand](#)
- [bkfilter](#)
- [callfun](#)
- [convert](#)
- [cosd](#)
- [cotd](#)
- [createDependentDummy](#)
- [csch](#)
- [dates](#)
- [denton](#)
- [diff](#)
- [egrowth](#)
- [eq](#)
- [expml](#)
- [floor](#)
- [getClassOfData](#)
- [growth](#)
- [head](#)
- [hpfilter](#)
- [iepcn](#)
- [ipcn](#)

- isempty
- kurtosis
- le
- log10
- lt
- mean
- min
- mpower
- mtimes
- not
- pca
- plus
- q2y
- reIndex
- realpow
- rlag
- secd
- setNan2Zero
- sind
- sqrt
- subAvg
- subsref
- tail
- tanh
- tonb_ts
- uplus
- window
- isfinite
- lag
- lead
- log1p
- mavg
- median
- minus
- mrdivide
- nan
- objSize
- pca_func
- pow2
- rand
- real
- realsqrt
- round
- sech
- sgrowth
- sinh
- std
- subSum
- sum
- tan
- times
- uminus
- var
- x12
- isnan
- ldivide
- log
- log2
- max
- merge2Series
- mldivide
- mstd
- ne
- or
- pcn
- power
- rdivide
- reallog
- ret
- sec
- set2Value
- sin
- skewness
- stdise
- subsasgn
- sumOAF
- tand
- toTseries
- undiff
- vertcat
- x12Census

► **abs ↑**

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the nb_math_ts

object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

► **acos** ↑

obj = acos(obj)

Description:

Inverse cosine, expressed in radians

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = acos(obj);
```

Written by Kenneth S. Paulsen

► **acosd** ↑

obj = acosd(obj)

Description:

Inverse cosine, expressed in degrees

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acosd(in);
```

Written by Andreas Haga Raavand

► **acosh** ↑

```
obj = acosh(obj)
```

Description:

Inverse hyperbolic cosine

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acosh(in);
```

Written by Andreas Haga Raavand

► **acot** ↑

```
obj = acot(obj)
```

Description:

Cotangent

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = acot(obj);
```

Written by Kenneth S. Paulsen

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

Inverse cotangent, expressed in degrees

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acotd(in);
```

Written by Andreas Haga Raavand

► **acoth** ↑

```
obj = acoth(obj)
```

Description:

Inverse hyperbolic cotangent

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acoth(in);
```

Written by Andreas Haga Raavand

► **acsc** ↑

```
obj = acsc(obj)
```

Description:

Inverse cosecant, expressed in radians

Input:

- obj : An object of class nb_math_ts

Output:

```
- obj : An object of class nb_math_ts
```

Examples:

```
obj = acsc(obj);
```

Written by Kenneth S. Paulsen

► **acsqd** ↑

```
obj = acsqd(obj)
```

Description:

Inverse cosecant, expressed in degrees

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acscd(in);
```

Written by Andreas Haga Raavand

► **acsch** ↑

```
obj = acsch(obj)
```

Description:

Inverse hyperbolic cosecant

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

► **and** ↑

```
a = and(a,b)
```

Description:

The and operator (&). Act element-wise on the data of the two nb_math_ts objects

Input:

- a : An object of class nb_math_ts

- b : An object of class nb_math_ts

Output:

- a : An object of class nb_math_ts. Where the and operator has evaluated all the data elements of the object. The data will be a logical matrix

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

► **append** ↑

```
obj = append(obj,aObj)
obj = append(obj,aObj,priority)
```

Description:

Append aObj to obj with a given priority.

Input:

- obj : A nObs1 x nVar x nPages nb_math_ts object.
- aObj : A nObs2 x nVar x nPages nb_math_ts object.
- priority : 'first' or 'second'. If 'first' all the observations from obj is used before the observations from aObj, otherwise it is reverse. 'first' is default.

Output:

- obj : A nObs x nVar x nPages nb_math_ts object.

Examples:

```
obj = nb_math_ts(zeros(10,2),'2018Q1');
aObj = nb_math_ts([nan(2,1),ones(2,1);ones(6,2)],'2020Q1');
new1 = append(obj,aObj);
new2 = append(obj,aObj,'second');
```

Written by Kenneth Sæterhagen Paulsen

► **appendFromAnotherPage** ↑

```
obj = appendFromAnotherPage(obj,page)
```

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_math_ts object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_math_ts object.

Examples:

```
obj          = nb_math_ts(zeros(10,2,4),'2018Q1');
obj(1:4,:,3:4) = nan;
new1         = appendFromAnotherPage(obj,1);
new2         = appendFromAnotherPage(obj,2);
```

Written by Kenneth Sætherhagen Paulsen

► **asec** ↑

```
obj = asec(obj)
```

Description:

Inverse secant, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

Inverse secant, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = asecd(in);
```

Written by Andreas Haga Raavand

► **asech** ↑

```
obj = asech(obj)
```

Description:

Inverse hyperbolic secant

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = asech(in);
```

Written by Andreas Haga Raavand

► **asin** ↑

```
obj = sin(obj)
```

Description:

Inverse sine, expressed in radians.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = sin(obj);
```

Written by Kenneth S. Paulsen

► **asind** ↑

```
obj = asind(obj)
```

Description:

Inverse sine, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

Inverse hyperbolic sine

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = asinh(in);
```

Written by Andreas Haga Raavand

► **atan ↑**

```
obj = atan(obj)
```

Description:

Inverse tangent, expressed in radians.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

► **atand ↑**

```
obj = atand(obj)
```

Description:

Inverse tangent, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = atand(in);
```

Written by Andreas Haga Raavand

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

Inverse hyperbolic tangent.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = atanh(in);
```

Written by Andreas Haga Raavand

► **avgOAF** ↑

```
obj = avgOAF(obj,sumFreq)
```

Description:

Take the average over a lower frequency. Ignoring nan values!

I.e. if frequency is quartely, this function can take the average over all the quarters of a year. And it will return the average over the year in all quarters of the that year

Input:

- obj : An object of class nb_math_ts
- sumFreq : The frequency to average over

Output:

- obj : An object of class nb_math_ts where the data is averaged over a lower frequency

Examples:

```
obj = nb_math_ts.rand('2012Q1',8);
obj = obj.avgOAF(1)
```

Written by Kenneth S. Paulsen

► **bkfilter** ↑

```
obj = bkfilter(obj,low,high)
```

Description:

Do band pass filtering of all the dataseries of the object.
(Returns the gap). Will strip nan values when calculating the filter

Input:

- obj : An object of class nb_math_ts
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_math_ts with the band pass filtered data.

Examples:

```
obj = bkfilter(obj,8,32);
obj = obj.bkfilter(8,32);
```

Written by Kenneth S. Paulsen

► **bkfilterls** ↑

```
obj = bkfilter1s(obj,low,high)
```

Description:

Do one sided band pass-filtering of all the dataseries of the object. (Returns the gap) Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_math_ts
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_math_ts with the one-sided band pass filtered data.

Examples:

```
obj = obj.bkfilter1s(12,24);  
obj = bkfilter1s(obj,12,24);
```

Written by Kenneth S. Paulsen

► **breakAdj ↑**

```
obj = breakAdj(obj,method,varargin)
```

Description:

The intent of this method is to set observations around break points to nan, and interpolate the observation at these break-points. I.e. if you have a level series that has a break, and want smooth growth rates. In this case you can calculate the growth rates of the level series, and then use this method to smooth out the growth rate at the break-points. Then you can use the inverse growth methods to transform back to a level series without breaks.

Input:

- obj : An object of class nb_math_ts
- method : The wanted interpolation method to use. See the nb_interpolate function.

Optional input:

- A set of nb_date objects or one line chars with the dates that should be set to nan and interpolated.

Output:

- obj : An object of class nb_math_ts.

See also:

[nb_interpolate](#), [growth](#), [egrowth](#), [pcn](#), [epcn](#), [igrowth](#), [iegrowth](#), [ipcn](#)
[iepcn](#)

Written by Kenneth S. Paulsen

► **callfun ↑**

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb_math_ts object(s). The data of the object is normally of class double, but may also be of class logical. Use nb_math_ts.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nb_math_ts.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- 'frequency' : If the operator should act on the data of the object, as it was on another frequency, you set this option to that frequency. Either 1 (yearly), 2 (semi-annually), 4 (quarterly), 12 (monthly), 52 (weekly) or 365 (daily).

Caution: It will be applied to the obj input as well as all optional inputs being a nb_math_ts object.

Caution: The method applied when converting to a higher frequency is 'none', and to a

lower frequency is 'discrete'. See the nb_math_ts.convert method for more on these methods.

- 'interpolateDate' : See the nb_math_ts.convert method for more on this option. Default is 'start'.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb_math_ts objects are converted to a matrix with elements of class double or logical.

Output:

- varargout : All output from the provided function which results in either a double or logical with the same size as obj input will be converted to a nb_math_ts object. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d      = nb_math_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)
```

See also:

[nb_dataSource.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

► **ceil ↑**

obj = ceil(obj)

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = ceil(in);
```

Written by Andreas Haga Raavand

► **conj** ↑

```
obj = conj(obj)
```

Description:

`conj(obj)` is the complex conjugate of the elements of `obj`.
For a complex element `x` of `obj`, `conj(x) = REAL(x) - i*IMAG(x)`.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts`

Examples:

```
out = conj(in);
```

Written by Andreas Haga Raavand

► **convert** ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the `nb_math_ts` object

Input:

- `obj` : An object of class `nb_math_ts`

- `freq` : The new frequency of the data. As an integer:

> 1	: yearly
> 2	: semi annually
> 4	: quarterly
> 12	: monthly
> 52	: weekly
> 365	: daily

- method : The method to use. Either:

From a high frequency to a low:

```
> 'average'      : Takes averages over the subperiods
> 'diffAverage' : Uses a weighted average, so that the
                  diff operators can be converted
                  correctly. E.g. when converting from monthly
                  to quarterly, and the level of the variable
                  of interest is aggregated to quarterly data
                  using averages, but you want to convert
                  monthly data on diff to quarterly. The
                  formula used is:
                    diff_Y(q) = 1/3*diff_Y(m) + 2/3*diff_Y(m-1)
                               + diff_Y(m-2) + 2/3*diff_Y(m-3)
                               + 1/3*diff_Y(m-4)
> 'diffSum'      : Same as 'diffAverage', but now the
                  aggregation in levels are done using a sum
                  instead.
> 'discrete'     : Takes last observation in each
                  subperiod (Default)
> 'first'        : Takes first observation in each
                  subperiod
> 'max'          : Takes maximal value over the subperiods
> 'min'          : Takes minimum value over the subperiods
> 'sum'          : Takes sums over the subperiods
```

From a low frequency to a high:

```
> 'linear'       : Linear interpolation
> 'cubic'        : Shape-preserving piecewise cubic
                  interpolation
> 'spline'       : Piecewise cubic spline interpolation
> 'none'         : No interpolation. All data in between
                  is given by nans (missing values)
                  (Default)
> 'fill'         : No interpolation. All periods of the
                  higher frequency get the same value as
                  that of the lower frequency.
> 'daverage'     : Uses the Denton 1971 method using that
                  the average of the intra low frequency
                  periods should preserve the average.
> 'dsum'         : Uses the Denton 1971 method using that
                  the sum of the intra low frequency
                  periods should preserve the sum.
```

Caution: This input must be given if you provide some
of the optional inputs.

Caution: All these option will by default use the
first period of the subperiods as base for
the conversion. Provide the optional input
'interpolateDate' if you want to set the base
to the end periods instead ('end').

Optional input:

- 'includeLast' : Give this string as input if you want to
include the last period, even if it is not

finished. Only when converting to lower frequencies

```
E.g: obj = obj.convert(1,'average',...
'includeLast');
```

- 'rename' : Changes the postfixes of the variables names.

E.g. If you covert from the frequency 12 to 4 the prefixes of the variable names changes from 'MUA' to 'QUA', if the prefix exist.

```
E.g: obj = obj.convert('yearly','average',...
'rename');
```

Optional input (...,'inputName', inputValue,...):

- 'interpolateDate' : The input after this input must either be 'start' or 'end'.

Date of interpolation. Where 'start' means to interpolate the start date of the periods of the old frequency, while 'end' uses the end date of the periods.

Default is 'start'.

Caution : Only when converting to higher frequency.

Caution : For weekly data, the 'dayOfWeek' options is added. See nb_ts.setDayOfWeek for more on this.

```
E.g: obj = obj.convert(365,'linear',...
'interpolateDate','end');
```

Output:

- obj : An nb_math_ts object converted to wanted frequency.

Examples:

```
obj = obj.convert(4,'average');
obj = obj.convert(365,'linear','interpolateDate','end');
obj = obj.convert(1,'average','includeLast');
obj = obj.convert(1,'average','rename');
```

Written by Kenneth S. Paulsen

► **corr ↑**

```
d = corr(obj)
```

Description:

Calculate the correlation of the timeseries stored in the nb_math_ts object

Input:

obj : An object of class nb_math_ts

Output:

d : A double with the wanted correlation

Examples:

```
corrMatrix = corr(obj)
```

Written by Kenneth S. Paulsen

► **cos** ↑

```
obj = cos(obj)
```

Description:

Cosine

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts.

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

`cosd(obj)` is the cosine of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts`

Examples:

```
out = cosd(in);
```

Written by Andreas Haga Raavand

► **cosh** ↑

```
obj = acsch(obj)
```

Description:

`cosh(obj)` is the hyperbolic cosine of the elements of `obj`.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts`

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

► **cot** ↑

```
obj = cot(obj)
```

Description:

Cotangent

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts.

Examples:

```
obj = cot(obj);
```

Written by Kenneth S. Paulsen

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

Cotangent, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = cotd(in);
```

Written by Andreas Haga Raavand

► **coth** ↑

```
obj = coth(obj)
```

Description:

Hyperbolic cotangent.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = coth(in);
```

Written by Andreas Haga Raavand

► **cov** ↑

```
obj = cov(obj)
```

Description:

Calculate the covariance of the timeseries stored in a nb_math_ts object

Input:

```
obj : An object of class nb_math_ts
```

Output:

d : A double with the wanted covariances

Written by Kenneth S. Paulsen

► **createDependentDummy** ↑

```
obj = createVarDummy(obj, condition)
obj = createVarDummy(obj, condition, scalar)
```

Description:

Creates a nb_math_ts object with dummy variables basted on the tested criterions.

Input:

- obj : An object of class nb_math_ts with size nObs x nVars x nPages.

- condition : Name of the added dummy variable.

- '<' : Less than.
- '>' : Greater than.
- '<=' : Less than or equal to.
- '>=' : Less than or equal to.
- '==' : Equal to.
- '~=': Not equal to.

- scalar : A scalar double. Default is 0.

Output:

- obj : An object of class nb_math_ts with size nObs x nVars x nPages.

Examples:

```
obj = nb_math_ts([1,2;-3,1;-1,3],'2012');  
obj = createDependentDummy(obj,'>',0);
```

See also:

[createDependentDummy](#)

Written by Kenneth Sæterhagen Paulsen

► **csc** ↑

obj = csc(obj)

Description:

Cosecant of argument in radians.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

Cosecant, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = cscd(in);
```

Written by Andreas Haga Raavand

► **csch** ↑

```
obj = csch(obj)
```

Description:

Hyperbolic cosecant

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

► **cumprod** ↑

```
obj = cumprod(obj,dim)
```

Description:

Cumulativ product

Input:

- obj : An object of class nb_math_ts
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : An object of class nb_math_ts where the data property of the object is now the cumulative product of the old objects data property.

Examples:

```
obj = cumprod(obj,1);
```

Written by Kenneth S. Paulsen

► **cumsum ↑**

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum

Input:

- obj : An object of class nb_math_ts
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An object of class nb_math_ts where the data property of the object is now the cumulative sum of the old objects data property.

Examples:

```
obj = cumsum(obj,1);
```

Written by Kenneth S. Paulsen

► **dates** ↑

```
allDates = dates(obj)
```

Description:

Get all the dates of the nb_math_ts object

Input:

- obj : An object of class nb_math_ts

Output:

- allDates : A cellstr with the dates of the object

Examples:

```
allDates = obj.dates();
```

Written by Kenneth S. Paulsen

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_math_ts object

- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_math_ts object.

See also:

[sgrowth](#), [subAvg](#).

Written by Tobias Ingebrigtsen

► **demeanMoving** ↑

```
obj = demeanMoving(data,backward)
obj = demeanMoving(data,backward,forward)
```

Description:

- Demeans data with a moving average process.

Input:

- obj : A nb_math_ts object
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average

Output:

- obj : A nb_math_ts object.

See also:

[nb_math_ts.demean](#)

Written by Kenneth Å!terhagen Paulsen

► **denton** ↑

```
obj = denton(obj,z,k,type,d)
```

Description:

The Denton method of transforming a series from low to high frequency.

See Denton (1971), Adjustment of Monthly or Quarterly Series to Annual Totals: An Approach Based on Quadratic Minimization.

Input:

- obj : A nobs x nvars x npage nb_math_ts object with the main variables to transform.
- z : A nobs*k x nvars x npage double with observations on the judgment.
- k : The number intra low frequency observations. If you have annual

data and want out quarterly data use 4.

- type : Either 'sum', 'average', 'first' or 'last'. 'average' is default.
- d : 1 : first differences, 2 : second differences.

Output:

- x : Output series as a nobs*k x nvars x npage nb_math_ts object.

See also:

[nb_math_ts.convert](#), [nb_denton](#)

Written by Kenneth Sæterhagen Paulsen

► **deptcat** ↑

obj = deptcat(a,b,varargin)

Description:

Depth concatenation (add pages of different nb_math_ts objects)
(No short hand notation)

Input:

- a : An object of class nb_math_ts
- b : An object of class nb_math_ts
- varargin : Optional number of nb_math_ts objects

Output:

obj : An object of class nb_math_ts, which will result from adding all the pages from the input objects.

Examples:

```
obj = deptcat(a,b);  
obj = deptcat(a,b,c);
```

Written by Kenneth S. Paulsen

► **detrend** ↑

```

obj          = detrend(obj,method)
obj          = detrend(obj,method,output,varargin)
[obj,trendObj] = detrend(obj,method,output,varargin)

```

Description:

Detrending nb_ts object.

Input:

- obj : As a nb_math_ts object.
- method : Either {'linear', 'linearls', 'mean', 'hpfilter', 'hpfilterls', 'bkfilter', 'bkfilterls', 'baiPerron' or 'exponentialsmoother'.
- output : If given as 'trend' the first output will be the trend output instead. 'Normal' is default.
- varargin : Additional inputs. When:
 - > 'exponentialsmoother' : One more input must be given. I.e. weight on previous value. See extrend function for more.
 - > 'hpfilter' or 'hpfilterls' : One more input must be given. I.e. lambda.
 - > 'bkfilter' or 'bkfilterls' : Two more inputs must be given. I.e. lowFreq and highFreq.
 - > 'baiPerron' : Two input may be given:
 - The first:
 - * 'bic' : Bayesian information criterion is used to select the breaks. Default.
 - * 'lwz' : The Liu, Wu and Zidek information criterion is used to select the breaks.
 - * integer : An integer with the number of breaks to identify.
 - The second:
 - * integer : The maximum number of breaks to allow when using a information criterion to choose the number of breaks. Default is 2.

Output:

- obj : As a nb_math_ts object with the detrended data.
- trendObj : As a nb_math_ts object with the trend.

Examples:

```

obj = nb_math_ts.rand('2012Q1',10,3);
obj = detrend(obj,'linear');
obj = detrend(obj,'hpfilter','','1600');
obj = detrend(obj,'bkfilter','','6,32');

```

See also:

[createShift](#)

Written by Kenneth Sæterhagen Paulsen

► **diff** ↑

```
obj = diff(obj)
obj = diff(obj, lags)
obj = diff(obj, lags, skipNaN)
```

Description:

Calculate diff, using the formula: $x(t) - x(t-lag)$ of all the timeseries of the nb_math_ts object.

Input:

- obj : An object of class nb_math_ts
- lags : The number of lags in the diff formula, default is 1.
- skipNaN : - 1 : Skip nan while using the diff operator. (E.g. when dealing with working days.)
 - 0 : Do not skip nan values. Default.

Output:

- obj : An nb_math_ts object with the calculated timeseries stored.

Examples:

```
obj = diff(obj);
obj = diff(obj, 4);
```

See also:

[nb_math_ts](#)

Written by Kenneth S. Paulsen

► **disp** ↑

```
disp(obj)
```

Description:

Display object (On the commandline)

Input:

obj : An object of class nb_math_ts

Output:

The object display on the command line. (When not ending the command with an semicolon)

Examples:

obj

Written by Kenneth S. Paulsen

► **double** ↑

dataOfObject = double(obj)

Description:

Get the data of the object as a double matrix

Input:

- obj : An object of class nb_math_ts

Output:

- dataOfObject : The data of the nb_math_ts object

Examples:

dataOfObject = double(obj)

Written by Kenneth S. Paulsen

► **egrowth** ↑

obj = egrowth(obj,lag,stripNaN)

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$ of all the timeseries of the nb_math_ts object.

Input:

- obj : An object of class nb_math_ts
- lag : The number of lags in the growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_math_ts object with the calculated timeseries stored.

Examples:

```
obj = egrowth(obj);
obj = egrowth(obj, 4);
```

Written by Kenneth S. Paulsen

► **empty** ↑

```
obj = empty(obj)
```

Description:

Empty the existing nb_math_ts object.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An empty nb_math_ts object

Examples:

```
obj = empty(obj)
```

Written by Kenneth S. Paulsen

► **epcn** ↑

```
obj = epcn(obj,lag,stripNaN)
```

Description:

Calculate exact percentage growth, using the formula:
 $100 * (x(t) - x(t-1)) / x(t-1)$ of all the timeseries of the
nb_math_ts object.

Input:

- obj : An object of class nb_math_ts
- lag : The number of lags in the growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_math_ts object with the calculated timeseries stored.

Examples:

```
obj = epcn(obj); % period-on-period growth
obj = epcn(obj,4); % 4-periods growth
```

Written by Kenneth S. Paulsen

► **eq ↑**

```
a = eq(a,b)
```

Description:

Test if the two objects are equal elemetswise and return a
nb_math_ts object where each element are 1 if true, otherwise 0.

The objects must have the same dimension

It is also possible to test each element of a object to a scalar

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- a : An nb_math_ts object where each element are 1 if the different data elemets are equal, otherwise 0.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

► **exp ↑**

```
obj = exp(obj)
```

Description:

Take exp of the data stored in the nb_math_ts object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts where the data
are equal to e raised to the data.

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

► **expand ↑**

```
obj = expand(obj,newStartDate,newEndDate,type,warningOff)
```

Description:

Expand the current nb_math_ts object with more dates and data

Input:

- obj : An object of class nb_math_ts
- newStartDate : The wanted new start date of the data.
- newEndDate : The wanted new end date of the data.
- type : Type of the appended data :

```

- 'nan'      : Expanded data is all nan (default)
- 'zeros'    : Expanded data is all zeros
- 'ones'     : Expanded data is all ones
- 'rand'     : Expanded data is all random numbers
- 'obs'      : Expand the data with first observation
                (before) or last observation after

- warningOff : Give 'off' to suppress warning if no expansion
               has taken place. Both 'newStartDate' and
               'newEndDate' is inside the window.

```

Output:

```
- obj          : An nb_math_ts object with expanded timespan
```

Examples:

```
obj = expand(obj,'1900Q1','2050Q1');
obj = expand(obj,'1900Q1','2050Q1','zeros');
```

Written by Kenneth S. Paulsen

► **expml ↑**

```
obj = expml(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, e to the `obj`.
`expml(obj)` computes $\exp(\text{obj}) - 1$, compensating for the roundoff in
 $\exp(\text{obj})$.
(For small real X , $\text{expml}(X)$ should be approximately X , whereas the
computed value of $\text{EXP}(X) - 1$ can be zero or have high relative error.)

Input:

```
- obj          : An object of class nb_math_ts
```

Output:

```
- obj          : An object of class nb_math_ts where the data are
               equal to  $e.^{\text{obj}.data} - 1$ 
```

Examples:

```
obj = expml(obj);
```

Written by Andreas Haga Raavand

► **extrapolate** ↑

```
obj = extrapolate(obj,toDate)
obj = extrapolate(obj,toDate,varargin)
```

Description:

Extrapolate the time-series.

Input:

- obj : An object of class nb_math_ts
- toDate : Extrapolate to the given date. If the date is before the end date of the given variable no changes will be made, and with no warning. Can also be a integer with the number of periods to extrapolate.

Optional inputs:

- 'alpha' : Significance level of ADF test for 'flow' or 'stock'.
- 'constant' : Give true to include constant in the AR models used to extrapolate the individual series. Default is true.
- 'freq' : The low frequency to fulfill the period for. E.g. if the object end at '2000M2', then '2000M3' is extrapolated so that all months of the 1st quarter of 2000 have values. toDate must be set to 'low'. Default is 1.
- 'method' :
 - > 'end' : Extrapolate series by using the last observation of each series. I.e. a random walk forecast. Default.
 - > 'ar' : Extrapolate each series individually using a fitted AR model.
- 'takeLog' : Take log before stock variables are extrapolated in first difference. true or false.
- 'type' : 'stock', 'flow' or 'test'. Default is 'flow'. 'stock' will extrapolate in (log) first difference. 'test' will use a ADF test to check if the variables are stationary ('flow') or not ('stock').

Output:

- obj : An nb_math_ts object with extrapolate series.

Examples:

```
obj = nb_math_ts.rand('2000M1',50,2,1);
obj = extrapolate(obj,'2004M3');
obj = extrapolate(obj,'2004M3','method','ar');
obj = extrapolate(obj,'low','method','ar','freq',4);
```

Written by Kenneth S. Paulsen

► **fillNaN** ↑

```
obj = fillNaN(obj)
obj = fillNaN(obj,date)
```

Description:

Fill in for nan values with last valid observation.

Input:

- obj : An object of class nb_math_ts.
- date : The end date of the returned dataset. Either a nb_date object or string with the date, or a number indicating how many periods back in time from the date today, i.e. 0 is today, -1 is past period and 1 is next period. Default is to use the end date of the object.

Caution : If the given date is before the end date of the data of the object no extrapolation is done.

Output:

- obj : An object of class nb_math_ts.

Examples:

```
d      = nb_math_ts.rand('2012Q1',10,3);
d(3,3) = nan;
d(6,1) = nan;
d2     = fillNaN(d);
```

Written by Kenneth SÃ¶terhagen Paulsen

► **floor** ↑

```
obj = floor(obj)
```

Description:

`floor(obj)` rounds the data elements of `obj` to the nearest integers towards minus infinity

Input:

- `obj` : An object of class `nb_data`

Output:

- `obj` : An object of class `nb_data`

Examples:

```
out = floor(in);
```

Written by Andreas Haga Raavand

► **ge ↑**

```
a = ge(a,b)
```

Description:

Test if the one object is greater or equal to the other object elemetswise and return a `nb_math_ts` object where each element are 1 if true, otherwise 0.

The objects must have the same dimension and variables

It is also possible to test each element of a object to a scalar

Input:

- `a` : An object of class `nb_math_ts` or a scalar
- `b` : An object of class `nb_math_ts` or a scalar

Output:

- `a` : An `nb_math_ts` object where each element are 1 if the data elemets of object `a` are greater or equal to the data elements of object `b`, otherwise 0.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get the properties of the object

Input:

- obj : An object
- propertyName :
 - > 'data' : The datz of the object as a double
 - > 'dim1' : Size of the first dimension
 - > 'dim2' : Size of the second dimension
 - > 'dim3' : Size of the third dimension
 - > 'endDate' : The end date of the object as string
 - > 'startDate' : The start date of the object as string

Output:

The property value of the object for the given property name

Examples:

```
dataAsDouble = get(obj,'data');
```

Written by Kenneth S. Paulsen

► **getClassOfData** ↑

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_math_ts.

Output:

- cl : Name of the class that the data of the object is stored as.

Written by Kenneth Sæterhagen Paulsen

► **getRealEndDate** ↑

```
realEndDate = getRealEndDate(obj,format)
```

Description:

Get the real end date of the nb_math_ts object. I.e the last observation which is not nan or infinite.

Input:

- obj : An object of class nb_math_ts

- format : The date format returned.

Return a string:

> 'xls'
> 'pprnorsk' or 'mprnorwegian'
> 'pprengelsk' or 'mprenglish'
> 'default' (otherwise)

Return a date object which is of a subclass of the nb_date class:

> 'nb_date'

Output:

- realEndDate : The last observation of the object which is not nan or infinite. As a string

Examples:

```
realEndDate = obj.getRealEndDate();
```

Written by Kenneth S. Paulsen

► **getRealStartDate** ↑

```
realStartDate = getRealStartDate(obj,format)
```

Description:

Get the real start date of the nb_math_ts object. I.e the first observation which is not nan or infinite.

Input:

- obj : An object of class nb_math_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'

Output:

- realStartDate : The first observation of the object which is not nan or infinite. As a string

Examples:

```
realEndDate = obj.getRealStartDate();
```

Written by Kenneth S. Paulsen

► growth ↑

```
obj = growth(obj)
obj = growth(obj,lag,stripNaN)
```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$ of all the timeseries of the nb_math_ts object.

Input:

- obj : An object of class nb_math_ts
- lag : The number of lags in the approx. growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_math_ts object with the calculated timeseries stored.

Examples:

```
obj = growth(obj);
obj = growth(obj,4);
```

Written by Kenneth S. Paulsen

► **gt** ↑

```
a = gt(a,b)
```

Description:

Test if the one object is greater then the other object elemetswise and return a nb_math_ts object where each element are 1 if true, otherwise 0.

The objects must have the same dimension and variables

It is also possible to test each element of a object to a scalar

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- a : An nb_math_ts object where each element are 1 if the data elemets of object a are greater then the data elements of object b, otherwise 0.

Examples:

```
obj = a > b;
obj = 2 > b;
obj = a > 2;
```

Written by Kenneth S. Paulsen

► **h21** ↑

```
obj = h21(obj,freq,type)
```

Description:

Will for each observation calculate the weighted cumulative sum over the last N periods (Including the present). The weights will depend on the frequency and the selected type. Examples will be given in the case of monthly data and freq = 4;

```

- 'levelSummed' : Y(q)      = Y(m) + Y(m-1) + Y(m-2)
- 'diffSummed' : Y(q)      = Y(m) + 2*Y(m-1) + 3*Y(m-2) +
                           2*Y(m-3) + Y(m-4)
- 'levelAverage' : Y(q)    = 1/3*(Y(m) + Y(m-1) + Y(m-2))
- 'diffAverage' : Y(q)    = 1/3*Y(m) + 2/3*Y(m-1) + Y(m-2) +
                           2/3*Y(m-3) + 1/3*Y(m-4)

```

Input:

- obj : An object of class nb_math_ts.
- freq : The low frequency to convert high frequency observations to.
- type : See description of method. Default is 'levelAverage'

Output:

- obj : An object of class nb_math_ts. Caution: The frequency of the nb_math_ts will not be converted. See nb_math_ts.convert in this case.

See also:

[nb_math_ts.convert](#)

Written by Kenneth S. Paulsen

► **head** ↑

`head(obj, nRows, page)`

Description:

Display the first n rows of a nb_math_ts object.

Input:

- obj : An nb_math_ts object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_math_ts](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► **horzcat** ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation ([a,b])

If the start or end dates differ, nan values will be appended

Input:

- a : An object of class nb_math_ts
- b : An object of class nb_math_ts
- varargin : Optional number of nb_math_ts objects

Output:

- obj : An object of class nb_math_ts with the input objects data are horizontally concatenated

Examples:

```
obj = [a,b];  
obj = [a,b,c];
```

Written by Kenneth S. Paulsen

► **horzcatfast** ↑

```
obj = horzcatfast(varargin)
```

Description:

Horizontal concatenation

The start or end dates cannot differ.

Input:

- varargin : Optional number of nb_math_ts objects

Output:

- obj : An object of class nb_math_ts with the input objects data are horizontally concatenated

See also
nb_math_ts.horzcat

Written by Kenneth S. Paulsen

► **hpfilter** ↑

```
obj = hpfilter(obj,lambda)
obj = hpfilter(obj,lambda,perc,fcstLen)
```

Description:

Do hp-filtering of all the dataseries of the nb_math_ts object.
(Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_math_ts
- lambda : The lambda of the hp-filter
- perc : Set to true to calculate the gap as (gap/trend)*100.
- fcstLen : The forecast length. Extrapolates the original series using the average of the last 4 periods. Default is 0.

Output:

- obj : An nb_math_ts object with the hp-filtered timeseries.

Examples:

```
gap = hpfilter(data,3000);
trend = data-gap;
```

Written by Kenneth S. Paulsen

► **hpfilterls** ↑

```
obj = hpfilterls(obj,lambda)
obj = hpfilterls(obj,lambda,perc,fcstLen)
```

Description:

Do one-sided hp-filtering of all the dataseries of the nb_math_ts object. (Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_math_ts
- lambda : The lambda of the hp-filter

- perc : Set to true to calculate the gap as (gap/trend)*100.
- fcstLen : The forecast length. Extrapolates the original series using the average of the last 4 periods. Default is 0.

Output:

- obj : An nb_math_ts object with the hp-filtered timeseries.

Examples:

```
gap = hpfilterls(data,3000);
trend = data-gap;
```

Written by Kenneth S. Paulsen

► **iegrowth** ↑

```
obj = iegrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact growth, i.e. the inverse method of the egrowth method of the nb_math_ts class

Input:

- obj : An object of class nb_math_ts
- InitialValues : A nb_math_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_math_ts object. If not provided 100 is default.
- Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_math_ts object with the indicies.

Examples:

```
obj = iegrowth(obj);
obj = iegrowth(obj,100);
obj = iegrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

► **iepcn** ↑

```
obj = iepcn(obj,initialValues,periods)
```

Description:

Construct indicies based on inital values and timeseries wich represent the series growth. Inverse of exact percentage growth, i.e. the inverse method of the epcn method of the nb_math_ts class

Input:

- obj : An object of class nb_math_ts
 - InitialValues : A nb_math_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_math_ts object. If not provided 100 is default.
- Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_math_ts object with the indicies.

Examples:

```
obj = iepcn(obj);
obj = iepcn(obj,100);
obj = iepcn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

► **igrowth** ↑

```
obj = igrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of log approx. growth, i.e. the inverse method of the growth method of the nb_math_ts class

Input:

- obj : An object of class nb_math_ts
- InitialValues : A nb_math_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_math_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_math_ts object with the indicies.

Examples:

```
obj = igrowth(obj);  
obj = igrowth(obj,100);  
obj = igrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

► **interpolate ↑**

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_math_ts object. Will discard leading and trailing nan values.

Input:

```
- data    : A nb_math_ts object.  
  
- method :  
  
- 'nearest'   : Nearest neighbor interpolation  
- 'linear'    : Linear interpolation. Default  
- 'spline'    : Piecewise cubic spline interpolation (SPLINE)  
- 'pchip'     : Shape-preserving piecewise cubic interpolation  
- 'cubic'     : Same as 'pchip'  
- 'v5cubic'   : The cubic interpolation from MATLAB 5, which  
                does not extrapolate and uses 'spline' if X is  
                not equally spaced.
```

Output:

```
- data : A nb_math_ts object with the interpolated data.
```

Written by Kenneth Sæterhagen Paulsen

► **ipcn ↑**

```
obj = ipcn(obj)  
obj = ipcn(obj,initialValues,periods,startAtNaN)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method of the nb_math_ts class

Input:

```
- obj           : An object of class nb_math_ts  
  
- InitialValues : A nb_math_ts object with a bigger window than obj,  
                  or a scalar or a double with the initial  
                  values of the indicies. Must be of the same  
                  size as the number of variables of the  
                  nb_math_ts object. If not provided 100 is  
                  default.  
  
                  Caution : If periods > 1 the initialValues input  
                             must be given, and has at least have  
                             periods number of rows.  
  
- periods      : The number of periods the initial series  
                  has been taken growth over.
```

Output:

```
- Output        : An nb_math_ts object with the indicies.
```

Examples:

```
obj = ipcn(obj);
obj = ipcn(obj,100);
obj = ipcn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_math_ts object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_math_ts

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Test if each element of an nb_math_ts object is finite.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_math_ts object is nan.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

► **kurtosis** ↑

```
obj = kurtosis(obj,flag,dim,outputType)
```

Description:

Calculate the kurtosis of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their kurtosis.

The output can also be set to be a double only consisting of the kurtosis values of the data.

Input:

- obj : An object of class nb_math_ts
- flag : > 0 : normalises by N-1 (Default)
 > 1 : normalises by N

Where N is the sample length.

- dim : The dimension to take the kurtosis over
- outputType : - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their kurtosis.
 - 'double': Get the kurtosis values as doubles, where each column of the double matches the same column of the data property of the nb_math_ts object. If the object consists of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are being set to their kurtosis. Or a double with the kurtosis values.

Examples:

```
obj = kurtosis(obj)  
same as  
obj = kurtosis(obj,0,0,'nb_math_ts')
```

Output as a double (matrix)

```
double = kurtosis(obj,0,0,'double')
```

Written by Kenneth S. Paulsen

► **lag ↑**

```
obj = lag(obj,periods)
```

Description:

Lag the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_math_ts
- periods : Lag the data with this number of periods. Default is 1 period.

Output:

- obj : An nb_math_ts object where the data lagged by number of periods given by the input periods.

Examples:

```
obj = lag(obj,1);
```

Written by Kenneth S. Paulsen

► **ldivide** ↑

```
obj = ldivide(a,b)
```

Description:

Left element-wise division (.\ \backslash). Same as right element-wise (./) division. See rdivide for more.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a.\b;
obj = 2.\b;
obj = a.\2;
```

See also:

[rdivide](#), [mldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

► **le** ↑

```
a = le(a,b)
```

Description:

Test if the one object is less or equal to the other object elemetswise and return a nb_math_ts object where each element are 1 if true, otherwise 0.

The objects must have the same dimension

It is also possible to test each element of a object to a scalar

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- a : An nb_math_ts object where each element are 1 if the data elemets of object a are less or equal to the data elements of object b, otherwise 0.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

► **lead** ↑

```
obj = lead(obj,periods)
```

Description:

Lead the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_math_ts
- periods : Lead the data with this number of periods. Default is 1 period.

Output:

- obj : An nb_math_ts object where the data leaded by number of periods given by the input periods.

Examples:

```
obj = lead(obj,1);
```

Written by Kenneth S. Paulsen

► **log** ↑

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_math_ts object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts where the data are on logs.

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

► **log10** ↑

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_math_ts object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

`log1p(obj)` computes $\text{LOG}(1+xi)$, where xi are the elements of `obj`. Complex results are produced if $xi < -1$.

For small real xi , `log1p(xi)` should be approximately xi , whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts`

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

► **log2** ↑

```
obj = log2(obj)
```

Description:

`log2(obj)` is the base 2 logarithm of the elements of `obj`.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts` where the data is the base 2 logs of `obj`.

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

► **lt** ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less then the other object elemetswise and return a nb_math_ts object where each element are 1 if true, otherwise 0.

The objects must have the same dimension

It is also possible to test each element of a object to a scalar

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- a : An nb_math_ts object where each element are 1 if the data elemets of object a are less then the data elements of object b, otherwise 0.

Examples:

```
obj = a < b;  
obj = 2 < b;  
obj = a < 2;
```

Written by Kenneth S. Paulsen

► **mavg** ↑

```
obj = mavg(obj,backward,forward,flag)
```

Description:

Taking moving avarage of all the timeseries of the nb_math_ts object

Input:

- obj : An object of class nb_math_ts
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average
- flag : If set to true the periods that does not have enough observations forward or backward should be set to nan. Default is false.

Output:

- obj : An nb_math_ts object storing the calculated moving average

Examples:

```
data = nb_math_ts(rand(50,2)*3,'2011Q1');
mAvg10 = mavg(data,9,0); % (10 year moving average)
```

Written by Kenneth S. Paulsen

► max ↑

```
obj = max(obj,dim,outputType)
```

Description:

Calculate the max values of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their max.

The output can also be set to be a double only consisting of the max values of the data.

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to find the max over
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their max.
 - 'double': Get the max values as doubles, where each column of the double matches the

same column of the data property of the nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are beeing set to their max. Or a double with the max values.

Examples:

```
obj = max(obj)
```

same as

```
obj = max(obj,1,'nb_math_ts')
```

Output as a double (matrix)

```
double = max(obj,1,'double')
```

Written by Kenneth S. Paulsen

► **mean ↑**

```
obj = mean(obj,dim,outputType)
```

Description:

Calculate the mean of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their mean.

The output can also be set to be a double only consisting of the mean values of the data.

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to take the mean over
- outputType : - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their mean.
- 'double': Get the mean values as doubles, where each column of the double matches the same column of the data property of the

nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are beeing set to their mean. Or a double with the mean values.

Examples:

```
obj = mean(obj)  
same as  
obj = mean(obj,1,'nb_math_ts')
```

Output as a double (matrix)

```
double = mean(obj,1,'double')
```

Written by Kenneth S. Paulsen

► **median** ↑

```
obj = median(obj,dim,outputType)
```

Description:

Calculate the median of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their median.

The output can also be set to be a double only consisting of the median values of the data.

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to take the median over
- outputType : - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their median.
- 'double': Get the median values as doubles, where each column of the double matches the same column of the data property of the

nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are being set to their median. Or a double with the median values.

Examples:

```
obj = median(obj)
```

same as

```
obj = median(obj,1,'nb_math_ts')
```

Output as a double (matrix)

```
double = median(obj,1,'double')
```

Written by Kenneth S. Paulsen

► **merge2Series ↑**

```
obj = merge2Series(obj,other,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_math_ts.
- other : An object of class nb_math_ts.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_math_ts.

Examples:

```
obj = nb_math_ts.rand('2012Q1',8,1,3);
other = nb_math_ts.rand('2012Q1',10,1,3);

obj1 = merge2Series(obj,other)
obj3 = merge2Series(obj,other,'diff','nLags',1)
obj4 = merge2Series(obj,other,'diff','nLags',4)
obj5 = merge2Series(obj,other,'level','date','2014Q1')
```

Written by Kenneth Sæterhagen Paulsen

► **min ↑**

```
obj = min(obj,dim,outputType)
```

Description:

Calculate the min values of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their min.

The output can also be set to be a double only consisting of the min values of the data.

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to find the min over
- outputType : - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their min.
- 'double': Get the min values as doubles, where each column of the double matches the same column of the data property of the nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are being set to their min. Or a double with the min values.

Examples:

```
obj = min(obj)
```

same as

```
obj = min(obj,1,'nb_math_ts')
```

Output as a double (matrix)

```
double = min(obj,1,'double')
```

Written by Kenneth S. Paulsen

► minus ↑

```
obj = minus(a,b)
```

Description:

Binary subtraction (-).

Subtract all elements of the first input objects data by the second input objects data pair-wise. The objects must be of the same sizes.

It is also possible to subtract data of the object by a scalar, or subtract a scalar by the objects data.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a-b;  
obj = 2-b;  
obj = a-2;
```

See also:

[plus](#)

Written by Kenneth S. Paulsen

► **mldivide** ↑

`obj = mldivide(a,b)`

Description:

Matrix left division (\). Only defined when dividing by a scalar or when a scalar are divide by the object(s data)

Input:

- `a` : An object of class nb_math_ts or a scalar
- `b` : An object of class nb_math_ts or a scalar

Output:

- `obj` : An object of class nb_math_ts with the calculated data stored

Examples:

`obj = 2\obj;`
`obj = obj\2;`

See also:

[rdivide](#), [ldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

► **mpower** ↑

`obj = power(a,b)`

Description:

Element-wise power (.^)

Makes it possible to raise the data of the obejct by a scalar, or raise a scalar by the objects data.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = 2^b;  
obj = a^2;
```

See also:

[power](#)

Written by Kenneth S. Paulsen

► **mrddivide** ↑

```
obj = mrddivide(a,b)
```

Description:

Matrix left division (/). Only defined when dividing by a scalar or when a scalar are divide by the object(s) data

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = 2/obj;  
obj = obj/2;
```

See also:

[rdivide](#), [ldivide](#), [mlddivide](#)

► **mstd** ↑

```
obj = mstd(obj,backward,forward)
```

Description:

Taking moving standard deviation of all the timeseries of the nb_math_ts object

Input:

- obj : An object of class nb_math_ts
- backward : Number of periods backward in time to calculate the moving std
- forward : Number of periods forward in time to calculate the moving std

Output:

- obj : An nb_math_ts object storing the calculated moving std

Examples:

```
data = nb_math_ts(rand(50,2)*3,'2011Q1');  
mstd10 = mstd(data,9,0); % (10 year moving std)
```

Written by Kenneth S. Paulsen

► **mtimes** ↑

```
obj = power(a,b)
```

Description:

Matrix multiplication (*) Only defined for multiplication with a scalar.

Makes it possible to multiply the data of the obejct by a scalar, or multiply a scalar by the objects data.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = 2*b;  
obj = a*2;
```

See also:

[times](#)

Written by Kenneth S. Paulsen

► **nan** ↑

```
obj = nb_math_ts.nan(start,obs,vars,pages)
```

Description:

Create an nb_math_ts object with all data set to nan.

Input:

- start : The start date of the created nb_math_ts object.
- obs : The number of observation of the created nb_math_ts object.
- vars : A scalar with the number of variables of the nb_math_ts object.
- pages : Number of pages of the nb_math_ts object.

Output:

- obj : An nb_math_ts object.

Examples:

```
obj = nb_math_ts.nan('2012Q1',2,2);  
obj = nb_math_ts.nan('2012Q1',2,2,10);
```

See also:

nb_math_ts

Written by Kenneth Sæterhagen Paulsen

► **ne ↑**

```
a = ne(a,b)
```

Description:

Test if the two objects are not equal elemetswise and return a nb_math_ts object where each element are 1 if true, otherwise 0.

The objects must have the same dimension

It is also possible to test each element of a object to a scalar

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- a : An nb_math_ts object where each element are 1 if the different data elemets are not equal, otherwise 0.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

► **not ↑**

```
obj = not(obj)
```

Description:

The not operator (~)

Input:

- a : An object of class nb_math_ts

Output:

- a : An object of class nb_math_ts where each element of the input object are equal to 1 if it was 0, otherwise 0. The data will be a logical matrix

Examples:

```
obj = ~obj;
```

Written by Kenneth S. Paulsen

► objSize ↑

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_math_ts
- dim : > 1 : Size of the first dimension
> 2 : Size of the second dimension
> 3 : Size of the third dimension

Output:

- varargout : > dim1 : Size of the first dimension
> dim2 : Size of the second dimension
> dim3 : Size of the third dimension

See the examples below.

Examples:

```
dim = objSize(obj)
```

Where dim is 1 x 2 double where the first element is the size of the first dimension and the second element is the size of the second dimension

```
[dim1, dim2] = objSize(obj)
```

```
[dim1, dim2, dim3] = objSize(obj)
```

```
dim1 = objSize(obj,1)
```

```
dim2 = objSize(obj,2)
```

```
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

► **or** ↑

```
a = or(a,b)
```

Description:

The or operator (|)

Will test elemetwise the data property of the two input objects and return 1 if both elemets are above 1, otherwise will return 0.

Input:

- a : An object of class nb_math_ts
- b : An object of class nb_math_ts

Output:

- a : An nb_math_ts object where the || operator have been used for the corresponding elements of the input objects data and the result are given by the output objects data property. The data will be a logical matrix

Examples:

```
a = a | b;
```

Written by Kenneth S. Paulsen

► **pca** ↑

```
F = pca(obj)
[F,LAMBDA,R,varF,expl,c,sigma,e,Z] = pca(obj,r,method,varargin)
```

Description:

Principal Component Analysis of the data of the object.

Input:

- obj : An object of class nb_math_ts.
- r : The number of principal component. If empty this number will be found by the Bai and Ng (2002) test, see the optional option crit below, i.e choose the CT part of the selection criterion; log(V(ii,F)) + CT. Where;

```

V(ii,F) = sum(diag(e_ii'*e_ii/T))/ii

and e_ii is the residual when ii factors are used

- method : Either 'svd' or 'eig'. Default is 'svd'.

    > 'svd' : Uses single value deomposition to construct the
               principal components. This is the numerically best
               way, as the 'eig' method could be very unprecise!

    > 'eig' : Uses the eigenvalue decomposition approach instead

```

Optional inputs:

```

- 'crit' : You can choose between the follwing selection criterion

    > 1: log(NT/NT1)*ii*NT1/NT;

    > 2: (NT1/NT)*log(min([N;T]))*ii;

    > 3: ii*log(GCT)/GCT;

    > 4: 2*ii/T;

    > 5: log(T)*ii/T;

    > 6: 2*ii*NT1/NT;

    > 7: log(NT)*ii*NT1/NT; (default)

where NT1 = N + T, GCT = min(N,T), NT = N*T and ii = 1:rMax

- 'rMax' : The maximal number of factors to test for. Must be less than
            or equal to N. Default is N.

- 'trans' : Give 'demean' if you want to demean the data in the matrix
            X. Default. Give 'standardize' if you want to standardise the
            data in the matrix X. Gice 'none' to drop any kind of
            transformation of the data in the matrix X.

Caution: To get back X when 'demean' is given you need to add
          the constant terms in c. I.e;

X = c(ones(T,1),1) + F*LAMBDA + e, e ~ N(0,R) (1')

To get back X when 'standardise' you need to add
the constant term and multiply with sigma. I.e.

X = c(ones(T,1),:) + F*LAMBDA.*sigma(ones(T,1),:) +
    eps, eps ~ N(0,R*) (1*)

Be aware that the eps differ from e, as e is the
residual from the standardised regression.

- 'unbalanced' : true or false. Set to true to allow for unbalanced
                  dataset. Default is false. If false all rows of the
                  dataset containing nan value are removed from
                  calculations.

```

Output:

- F : The principal component, as a nb_math_ts object.
- LAMBDA : The estimated loadings, as a double
- R : The covariance matrix of the residual in (1), as a double.
- varF : As a double. Each column will be the variance of the corresponding column of F.
- expl : The percentage of the total variance explained by each principal component. As a 1 x r double
- c : Constant in the factor equation. double with size 1 x nvar.
- sigma : See 'trans'. double with size 1 x nvar.
- e : Residual of the equation (1) above. As a nb_math_ts object.
- Z : Normalized (and rebalanced) version of X.

See also:[nb_pca](#)

Written by Kenneth Sæterhagen Paulsen

► pca_func ↑

```
factors = pca_func(varargin)
```

Description:

Principal Component of selected number of series represented by a set of nb_math_ts.

Optional input:

- varargin : Optional number of nb_math_ts objects.
- 'numFactors' : Determine number of Factors to report. Standard is first factor only.
- 'unbalanced' : true or false.
- 'whichFactor' : Select the factor to return.

Output:

- factors : The principal component(s) as a nb_math_ts object.

See also:

[nb_math_ts.pca](#)

Written by Kenneth Sætherhagen Paulsen

► **pcn ↑**

obj = pcn(obj,lag,stripNaN)

Description:

Calculate log approximated percentage growth. Using the formula
 $(\log(t+lag) - \log(t)) * 100$

Input:

- obj : An object of class nb_math_ts
- lag : The number of lags. Default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_math_ts object with the log approximated percentage growth data stored.

Examples:

```
obj = pcn(obj); % period-on-period log approx. growth
obj = pcn(obj,4); % 4-periods log approx. growth
```

See also:

[epcn](#)

Written by Kenneth S. Paulsen

► **plus ↑**

obj = plus(a,b)

Description:

Binary addition (+).

Added all elements of the two input objects data pair-wise.
The objects must be of the same sizes.

It is also possible to add a scalar to the data of the object.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a+b;  
obj = 2+b;  
obj = a+2;
```

See also:

[minus](#)

Written by Kenneth S. Paulsen

► **pow2** ↑

```
obj = pow2(obj)
```

Description:

pow2(obj) raises the number 2 to the power of the elements of obj

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts where the data are the number 2 raised to the power of the elements of the original object.

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

► **power** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^)

All elements of the first input objects data is raised by the second input objects data pair-wise. The objects must be of the same sizes.

It is also possible to raise the data of the object by a scalar, or raise a scalar by the objects data, but it is recommended to use mpower (^) instead.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a.^b;
obj = 2.^b;
obj = a.^2;
```

See also:

[mpower](#)

Written by Kenneth S. Paulsen

► **q2y ↑**

```
obj = q2y(obj)
```

Description:

Will for each quarter calculate the cumulative sum over the last 4 quarter (Including the present). The frequency of the return object will be quarterly.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An nb_math_ts object where the data are the sum over the last 4 quarters.

Written by Kenneth S. Paulsen

► **rand** ↑

```
obj = nb_math_ts.rand(start,obs,vars,pages,dist)
```

Description:

Create an nb_math_ts object with all data set to random number.

Input:

- start : The start date of the created nb_math_ts object.
- obs : The number of observation of the created nb_math_ts object.
- vars : A scalar with the number of variables of the nb_math_ts object.
- pages : Number of pages of the nb_math_ts object.
- dist : A nb_distribution object to draw from.

Output:

- obj : An nb_math_ts object.

Examples:

```
obj = nb_math_ts.rand('2012Q1',2,2);
obj = nb_math_ts.rand('2012Q1',2,2,10,nb_distribution('type','normal'));
```

See also:

[nb_math_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

```
obj = rdivide(a,b)
```

Description:

Right element-wise division (./)

Divide all elements of the first input objects data by the second input objects data pair-wise. The objects must be of the same sizes.

It is also possible to divide data of the obejct by a scalar, or divide a scalar by the objects data, but it is recommended to use mrdivide instead.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a./b;
obj = 2./b;
obj = a./2;
```

See also:

[mrdivide](#), [mldivide](#), [ldivide](#)

Written by Kenneth S. Paulsen

► **reIndex** ↑

```
obj = reIndex(obj,date)
obj = reIndex(obj,date,value)
```

Description:

Reindex the data of the object to a new date. It is also possible to reindex the timeseries to a date with lower frequency. E.g. reindex quarterly data to be on average 100 in a given year.

Input:

- obj : An object of class nb_math_ts
- date : The date at which the data should be reindexed to.
- value : The value to re-index the data to. Default is 100

Output:

- obj : An object of class nb_math_ts where all the timeseries are reindexed to value at the given date.

Examples:

```
obj = reIndex(obj,'2012Q2')
```

Written by Kenneth S. Paulsen

► **real** ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part of the elements in the nb_math_ts object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts where the data are the real part of the elements of the original object.

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

► **reallog** ↑

```
obj = reallog(obj)
```

Description:

Take the natural logarithm of the data stored in the object

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts where the data are the natural logarithms of the elements of the original object.

Examples:

```
obj = reallog(obj);
```

Written by Kenneth S. Paulsen

► **realpow** ↑

```
obj = power(a,b)
```

Description:

Real power.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts

See also:

[mpower](#), [power](#)

Written by Kenneth S. Paulsen

► **realsqrt** ↑

```
obj = realsqrt(obj)
```

Description:

`realsqrt(obj)` is the square root of the elements of `obj`.
An error is produced if `obj` contains negative elements.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = realsqrt(in);
```

Written by Andreas Haga Raavand

► **ret ↑**

```
obj = ret(obj)
obj = ret(obj,nlag)
obj = ret(obj,nlag,skipNaN)
```

Description:

Calculate return, using the formula: $x(t)/x(t-lag)$ of all the timeseries of the nb_math_ts object.

Input:

- obj : An object of class nb_math_ts
- nlag : The number of lags in the return formula,
default is 1.
- skipNaN : - 1 : Skip nan while using the ret operator. (E.g.
when dealing with working days.)
- 0 : Do not skip nan values. Default.

Output:

- obj : An nb_math_ts object with the calculated timeseries stored.

Examples:

```
obj = ret(obj);
obj = ret(obj,4);
```

See also:

[nb_math_ts](#)

Written by Kenneth S. Paulsen

► **rlag ↑**

```
obj = rlag(obj, lags)
obj = rlag(obj, lags, periods)
```

Description:

Roll a pattern in the data forward with a given lag.

Input:

- obj : An object of class nb_math_ts
- lags : The number of lags
- periods : The extrapolated periods. The end date of the object will be the current end date plus these number of periods. Be aware that trailing nan values in the data will be filled in for even if periods == 0, which is the default.

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj1 = nb_math_ts([1,1;2,2;3,3;4,4;1,nan],'2012Q1');
obj2 = rlag(obj1,4,0);
obj2 = rlag(obj1,4,3);
```

See also:

[nb_rlag](#)

Written by Kenneth Sæterhagen Paulsen

► **round** ↑

```
obj = round(obj,value,startDate,endDate,pages)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_math_ts object
- value : The rounding value. As a double. Default is 1.
- startDate : The start date of the rounding. As a date string or a nb_date object. Can be empty. Default is the start date

of the data.

- endDate : The end date of the rounding. As a date string or a nb_date object. Can be empty. Default is the end date of the data.
- variables : The variables to round. Can be empty. Default is to round all variables.

Output:

- obj : A nb_math_ts object with the rounded numbers.

Written by Kenneth Sæterhagen Paulsen

► **sec ↑**

obj = sec(obj)

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

obj = sec(obj);

Written by Kenneth S. Paulsen

► **secd ↑**

obj = secd(obj)

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

► **set2Value** ↑

```
obj = set2Value(obj,func,value)
```

Description:

Set all values applying to the restriction given by the input func to the provided value.

Input:

- obj : An object of class nb_math_ts.
- func : A function handle that takes in a double with size M x N x P and return a logical with same size.
- value : A scalar double.

Output:

- obj : An object of class nb_math_ts.

Examples:

```
obj = nb_math_ts.rand('2000',10,1) - 0.5;
obj = set2Value(obj,@(x)x<0,0)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **setNan2Zero** ↑

```
obj = setNan2Zero(obj)
```

Description:

Set all nan values of the data of the object to 0.

Input:

- obj : An object of class nb_math_ts.

Output:

- obj : An object of class nb_math_ts.

Written by Kenneth SÃ¶terhagen Paulsen

► **sgrowth** ↑

```
obj = sgrowth(obj,horizon)
```

Description:

Calculates smoothed 12 or 6 months growth.

```
x(*) = ((x13+x14+x15)/(x1+x2+x3)-1)*100
```

```
x(*) = ((x7+x8+x9)/(x1+x2+x3)-1)*100
```

Note: Works only for monthly data.

Based on Anne Sofie Jores data transformation code.

Input:

- obj : A nb_math_ts object
- horizon : Either 1 or 2, depending on the frequency you want your growth to be calculated over. 1 is annual and 2 is semi-annual.

Output:

- out : A nb_math_ts object.

Examples:

See also:

nb_sgrowth

Written by Tobias Ingebrigtsen

► **sin ↑**

obj = sin(obj)

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts.

Examples:

obj = sin(obj);

Written by Kenneth S. Paulsen

► **sind ↑**

obj = sind(obj)

Description:

Sine, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = sind(in);
```

Written by Andreas Haga Raavand

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

Hyperbolic sine.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = sinh(in);
```

Written by Andreas Haga Raavand

► **skewness** ↑

```
obj = skewness(obj,flag,dim,outputType)
```

Description:

Calculate the skewness of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their skewness.

The output can also be set to be a double only consisting of the skewness values of the data.

Input:

- obj : An object of class nb_math_ts
- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N
 - Where N is the sample length.
- dim : The dimension to take the skewness over
- outputType : - 'nb_math_ts': The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are being set to their skewness.
 - 'double': Get the skewness values as doubles, where each column of the double matches the same column of the data property of the nb_math_ts object. If the object consists of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are being set to their skewness. Or a double with the skewness values.

Examples:

```
obj = skewness(obj)
```

same as

```
obj = skewness(obj,0,0,'nb_math_ts')
```

Output as a double (matrix)

```
double = skewness(obj,0,0,'double')
```

Written by Kenneth S. Paulsen

► **sqrt ↑**

```
obj = sqrt(obj)
```

Description:

`sqrt(obj)` is the square root of the elements of `obj`. Complex results are produced for non-positive elements.

Input:

- `obj` : An object of class `nb_math_ts`

Output:

- `obj` : An object of class `nb_math_ts`

Examples:

```
out = sqrt(in);
```

Written by Andreas Haga Raavand

► **std ↑**

```
obj = std(obj,flag,dim,outputType)
```

Description:

Calculate the std of each timeseries. The result will be an object of class `nb_math_ts` where all the non-nan values of all the timeseries are being set to their std.

The output can also be set to be a double only consisting of the std values of the data.

Input:

- `obj` : An object of class `nb_math_ts`

- `flag` : > 0 : normalises by N-1 (Default)
 > 1 : normalises by N

Where N is the sample length.

- `dim` : The dimension to take the std over

- `outputType` : - 'nb_math_ts': The result will be an object of class `nb_math_ts` where all the non-nan values of all the timeseries are being set to their std.

- 'double': Get the std values as doubles, where each column of the double matches the

same column of the data property of the nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are beeing set to their std. Or a double with the std values.

Examples:

```
obj = std(obj)
```

same as

```
obj = std(obj,0,0,'nb_math_ts')
```

Output as a double (matrix)

```
double = std(obj,0,0,'double')
```

Written by Kenneth S. Paulsen

► **stdise** ↑

```
obj = stdise(obj,flag)
```

Description:

Standardise data of the object by subtracting mean and dividing by std deviation.

Input :

- obj : An object of class nb_math_ts
- flag : - 0 : normalises by N-1 (Default)
- 1 : normalises by N

Where N is the sample length.

Output:

- obj : An nb_math_ts object with the standardised data

Examples:

```
obj = stdise(obj);
```

Written by Kenneth S. Paulsen

► **subAvg ↑**

```
obj = subAvg(obj,k)
obj = subAvg(obj,k,w)
```

Description:

Will for the last k periods (including the present) calculate the cumulative sum and then divide by the amount of periods, which gives you the average over those periods.

Input:

- obj : An object of class nb_math_ts.
- k : Lets you choose what frequency you want to calculate the average over. As a double. E.g. 4 if you have quarterly data and want to calculate the average over the last 4 quarters.

Output:

- obj : An nb_math_ts object where the data are the average over the last k periods.

Examples:

```
obj = subAvg(obj,k)
```

See also:

[growth](#), [q2y](#), [pcn](#), [subSum](#)

Written by Tobias Ingebrigtsen

► **subSum ↑**

```
obj = subSum(obj,k)
obj = subSum(obj,k,w)
```

Description:

Calculates the cumulative sum over the last k periods (including the present period).

Input:

- obj : An object of class nb_math_ts.
- k : Lets you choose what frequency you want to calculate the cumulative sum over. As a double. E.g. 4 if you have quarterly data and want to calculate the average over the last 4 quarters.
- w : Weights applies to the k periods to sum over. Default is equal weights! As a double vector with length k.

Output:

- obj : An object of class nb_math_ts.

Examples:

```
obj = subSum(obj,k)
```

See also:

pcn, growth, subAvg

Written by Tobias Ingebrigtsen

► subsasgn ↑

```
varargout = subsasgn(obj,s,b)
```

Description:

Makes it possible to do subscripted assignment of the data of an object

Input:

- obj : An object of class nb_math_ts
- s : The same input as when you do subscripted assignment one a double matrix
- b : The assign data at the given index s, as a double or nb_math_ts object (disregard time dimension in this case!).

Output:

- obj : The assign data property setted of the given object.

Examples:

```
obj      = nb_math_ts.rand('2012',10,3);
obj(1:3,1) = [1;2;3]
obj =
    'Time'      'Var1'
    '2012'      [ 1]
    '2013'      [ 2]
    '2014'      [ 3]
```

Written by Kenneth S. Paulsen

► **subsref ↑**

```
varargout = subsref(obj,s)
```

Description:

Makes it possible to do subscripted reference of the data of an nb_math_ts object. See the examples for more

Using the method window, just using indexing instead

Input:

- obj : An object of class nb_math_ts
- s : A structure on have to index the object

Output:

- obj : The indexed object.

Examples:

```
obj = obj(1,2:4,1)
```

same as

```
obj = obj(1,2:4)
```

Be aware that that obj(1,:) will work but not obj(1,2:end)

```
obj = obj('1994Q1')
```

same as

```
obj.window('1994Q1','1994Q1')
```

```
obj = obj('1994Q1','1994Q4')
```

same as

```
obj.window('1994Q1','1994Q4')

obj = obj('1994Q1','1994Q4',1:2)

same as

obj.window('1994Q1','1994Q4',1:2)
```

Written by Kenneth S. Paulsen

► **sum ↑**

```
obj = sum(obj,dim)
```

Description:

Takes the sum of the object data in the wanted dimension

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to sum over, returns:
 - dim = 1 : An nb_math_ts object with all the variables set to their cross observation sum
 - dim = 2: An nb_math_ts object with all variables set to their cross variable sum (Take the sum over the variables)
 - dim = 3: An nb_math_ts with only one page, representing the sum over all pages.

Caution : All sums ignore nan values. (If all values is not nan.)

Output:

- obj : An nb_math_ts object representing the sum.
(Keeps the dimension)

Examples:

```
obj = sum(obj,1)
obj = sum(obj,2)
obj = sum(obj,3)
```

Written by Kenneth S. Paulsen

► **sumOAF** ↑

```
obj = sumOAF(obj,sumFreq)
```

Description:

Take the sum over a lower frequency.

I.e. if frequency is quartely, this function can sum over all the quarters of a year. And it will return the sum over the year in all quarters of the that year

Input:

- obj : An object of class nb_math_ts
- sumFreq : The frequency to sum over

Output:

- obj : An object of class nb_math_ts where the data is summed over a lower frequency

Examples:

```
obj = nb_math_ts.rand('2012Q1',8);
obj = obj.sumOAF(1)
```

Written by Kenneth S. Paulsen

► **tail** ↑

```
tail(obj,nRows,page)
```

Description:

Display the last n rows of a nb_math_ts object.

Input:

- obj : An nb_math_ts object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_math_ts](#), [window](#), [head](#)

► **tan** ↑

```
obj = tan(obj)
```

Description:

Tangent

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
obj = tan(obj);
```

Written by Kenneth S. Paulsen

► **tand** ↑

```
obj = tand(obj)
```

Description:

Tangent, expressed in degrees.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = tand(in);
```

Written by Andreas Haga Raavand

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

Hyperbolic tangent.

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An object of class nb_math_ts

Examples:

```
out = tanh(in);
```

Written by Andreas Haga Raavand

► **times** ↑

```
obj = times(a,b)
```

Description:

Element-wise multiplication (.*)

All elements of the first input objects data is multiplied by the second input objects data pair-wise. The objects must be of the same sizes.

It is also possible to multiply the data of the obejct by a scalar, or multiply a scalar by the objects data, but it is recommended to use mtimes (*) instead.

Input:

- a : An object of class nb_math_ts or a scalar
- b : An object of class nb_math_ts or a scalar

Output:

- obj : An object of class nb_math_ts with the calculated data stored

Examples:

```
obj = a.*b;
obj = 2.*b;
obj = a.*2;
```

See also:

[mtimes](#)

Written by Kenneth S. Paulsen

► **toTseries** ↑

```
tseries_DB = toTseries(obj)
```

Description:

Transform from an nb_math_ts object to an tseries object (IRIS)

To use this function you need to add the IRIS package to the MATLAB path.

Input:

- obj : An object of class nb_math_ts

Output:

- tseries_DB : An object of class tseries

Examples:

```
tseries_DB = obj.toTseries();
```

Written by Kenneth S. Paulsen

► **tonb_ts** ↑

```
obj = tonb_ts(obj,variables,dataName)
```

Description:

Convert the nb_math_ts object to an nb_ts object

Input:

- obj : An object of class nb_math_ts
- variables : The variable names of the created nb_ts object.
- dataName : The name of the dataset(s) of the created nb_ts object. Either a string or an cellstr. When given as cellstr it must match the number of pages of the provided nb_math_ts object. If not given default name(s) will be used.

Output:

- obj : An object of class nb_ts

Examples:

```
obj = nb_math_ts([2,2;2,2],'2012Q1');
obj1 = tonb_ts(obj,{'Var1','Var2'},'test')
obj2 = tonb_ts(obj,{'Var1','Var2'})
```

See also:

[nb_ts](#)

Written by Kenneth S. Paulsen

► **uminus** ↑

obj = uminus(obj)

Description

Unary minus

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An nb_math_ts object where all the data are the unary minus of the input objects data

Examples:

obj = -obj;

Written by Kenneth S. Paulsen

► **undiff** ↑

```
obj = undiff(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series diff. Inverse of the diff method.

Input:

- obj : An object of class nb_math_ts
- InitialValues : A nb_math_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_math_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken diff over.

Output:

- Output : A nb_math_ts object with the indicies.

Examples:

```
obj = undiff(obj);  
obj = undiff(obj,100);  
obj = undiff(obj,[78,89]);
```

See also
diff

Written by Kenneth S. Paulsen

► uplus ↑

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_math_ts

Output:

- obj : An nb_math_ts object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

► var ↑

```
obj = var(obj,dim,outputType)
```

Description:

Calculate the variance of each timeseries. The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their var.

The output can also be set to be a double only consisting of the var values of the data.

Input:

- obj : An object of class nb_math_ts
- dim : The dimension to take the var over
- outputType :
 - 'nb_math_ts' : The result will be an object of class nb_math_ts where all the non-nan values of all the timeseries are beeing set to their var.
 - 'double' : Get the var values as doubles, where each column of the double matches the same column of the data property of the nb_math_ts object. If the object consist of more pages the double will also consist of more pages.

Output:

- obj : The object itself where all the non-nan values of all the timeseries are beeing set to their var. Or a double with the var values.

Examples:

```
obj = var(obj)  
      same as  
  
obj = var(obj,0,0,'nb_math_ts')  
  
Output as a double (matrix)  
  
double = var(obj,0,0,'double')  
  
Written by Kenneth S. Paulsen
```

► **vertcat** ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_math_ts
- b : An object of class nb_math_ts
- varargin : Optional numbers of objects of class nb_math_ts

Output:

- a : An nb_math_ts object where the data from the different objects are appended to each other.

Examples:

```
obj = nb_math_ts(ones(2,1),'2012Q1');  
aObj = nb_math_ts(ones(2,1),'2012Q3');  
m = [obj;aObj]  
  
m =  
  
'2012Q1' [ 1]  
'2012Q2' [ 1]  
'2012Q3' [ 1]  
'2012Q4' [ 1]
```

Written by Kenneth S. Paulsen

► **window** ↑

```
obj = window(obj,startDateWin,endDateWin, pages)
```

Description:

Narrow down window of the data of the nb_math_ts object

Input:

- obj : An object of class nb_math_ts
- startDateWin : The new wanted start date of the data of the object. Must be a string on the format; 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the start date of the data.
Can also be an object which is a subclass of the nb_date class.
- endDateWin : The new wanted end date of the data of the object. Must be a string on the format. 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the end date of the data.
Can also be an object which is a subclass of the nb_date class.
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty all pages is kept.
I.e. obj = obj.window('', '', {}, 1:3)

Output:

- obj : An nb_math_ts object where the datasets of the object is narrowed down to the specified window.

Examples:

```
obj = nb_math_ts(ones(10,2,2),'2012Q1');
obj = obj.window('2012Q2')
obj = obj.window('', '2013Q4')
obj = obj.window('', '', 1)
```

See also:

Written by Kenneth S. Paulsen

► x12 ↑

```
varargout = x12(obj,range,varargin)
```

Description:

Seasonally adjust data of an nb_math_ts object, with the use of the x12 method of the tseries class provided by the IRIS package.

To use this function you need to add the IRIS package to the MATLAB path.

Input:

- obj : Input data that will seasonally adjusted or filtered by the Census X12 Arima. Must be an nb_math_ts object.

- range : Date range on which the X12 will be run; if not specified or Inf the entire available range will be used. Must be numeric.

Optional input (..., 'propertyName', PropertyValue, ...):

- 'backcast' : Run a backcast based on the fitted ARIMA model for this number of periods back to improve on the seasonal adjustment; see help on the 'x11' specs in the X12-ARIMA manual. The backcast is included in the output argument obj. Must be a scalar.

- 'cleanup' : Delete temporary X12 files when done; the temporary files are named iris_x12a.*. {'true'} | 'false'

- 'log' : Logarithmise the input data before, and de-logarithmise the output data back after, running x12. {'true'} | 'false'

- 'forecast' : Run a forecast based on the fitted ARIMA model for this number of periods ahead to improve on the seasonal adjustment; see help on the x11 specs in the X12-ARIMA manual. The forecast is included in the output argument obj. Must be a scalar.

- 'display' : Display X12 output messages in command window; if false the messages will be saved in a TXT file. {'true'} | 'false'

- 'dummy' : Dummy variable or variables (in case of a multivariate tseries object) used in X12-ARIMA regression; the dummy variables can also include values for forecasts and backcasts if you request them. Either an tseries or empty.

- 'dummyType' : Type of dummy; see the X12-ARIMA IRIS documentation. Either 'ao' | {'holiday'} | 'td'

- 'mode' : Seasonal adjustment mode (see help on the x11 specs in the IRIS X12-ARIMA manual); 'auto' means that series with only positive or only negative

```

numbers will be adjusted in the 'mult'
(multiplicative) mode, while series with combined
positive and negative numbers in the 'add'
(additive) mode. {'auto'} | 'add' | 'logadd' |
'mult' | 'pseudoadd' | 'sign'

- 'maxIter' : Maximum number of iterations for the X12
estimation procedure. See help on the estimation
specs in the IRIS X12-ARIMA manual. Must be
numeric. Default is 1500.

- 'maxOrder' : A 1-by-2 vector with maximum order for the
regular ARMA model (can be 1, 2, 3, or 4) and
maximum order for the seasonal ARMA model (can be
1 or 2). See help on the automdl specs in the
IRIS X12-ARIMA manual. Must be numeric. Default
is [1,2]. 

- 'missing' : Allow for in-sample missing observations, and
fill in values predicted by an estimated ARIMA
process; if false, the seasonal adjustment will
not run and a warning will be thrown. 'true' |
{'false'}

- 'output' : List of requested output data; the cellstr or
comma-separated list can combine 'IR' for the
irregular component, 'seasadj' for the final
seasonally adjusted series, 'SF' for seasonal
factors, 'trendcycle' for the trend-cycle, and
'MV' for the original series with missing
observations replaced with ARIMA estimates. See
also help on the x11 specs in the IRIS X12-ARIMA
manual. char | cellstr | {'seasadj'} ]

- 'specFile' : Name of the X12-ARIMA spec file; if 'default' the
IRIS default spec file will be used, see
description. Must be a char. Default is 'default'

- 'tdays' : Correct for the number of trading days. See help
on the x11regression specs in the IRIS X12-ARIMA
manual. 'true' | {'false'}

- 'tolerance' : Convergence tolerance for the X12 estimation
procedure. See help on the estimation specs in
the IRIS X12-ARIMA manual. Must be numeric.
Default is 1e-5.

```

Output:

See the IRIS X12-ARIMA manual. Same outputs as the x12 method of the tseries class. The only difference is that the tseries outputs are instead nb_math_ts objects.

Examples:

```
[obj,~,err] = x12(obj,inf,'output','seasadj');
```

► **x12Census** ↑

```
obj = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj,varargin)
```

Description:

Do x12 Census adjustement to an nb_math_ts object using x12awin.exe.

This is a file copied from the IRIS Toolbox an adapted to the NB Toolbox. (I.e. using nb_math_ts instead of tseries)

x12awin.exe is a program developed by:

U. S. Department of Commerce, U. S. Census Bureau

X-12-ARIMA quarterly seasonal adjustment Method,
Release Version 0.3 Build 192

This method modifies the X-11 variant of Census Method II
by J. Shiskin A.H. Young and J.C. Musgrave of February, 1967.
and the X-11-ARIMA program based on the methodological research
developed by Estela Bee Dagum, Chief of the Seasonal Adjustment
and Time Series Staff of Statistics Canada, September, 1979.

This version of X-12-ARIMA includes an automatic ARIMA model
selection procedure based largely on the procedure of Gomez and
Maravall (1998) as implemented in TRAMO (1996).

Caution : The object must be of quarterly or monthly frequency!!

Input:

- obj : Input data that will seasonally adjusted or filtered
by the Census X12 Arima. Must be an nb_math_ts object.

Optional input (..., 'propertyName', PropertyValue, ...):

- 'backcast' : Run a backcast based on the fitted ARIMA model
for this number of periods back to improve on the
seasonal adjustment. The backcast is included
in the output argument x. Must be a scalar.

- 'log' : Logarithmise the input data before, and
de-logarithmise the output data back after,
running x12. {1} | 0

- 'forecast' : Run a forecast based on the fitted ARIMA model
for this number of periods ahead to improve on
the seasonal adjustment. The forecast is included
in the output argument x. Must be a scalar.

- 'display' : Display X12 output messages in command window.
{0} | 1. Default is not.
- 'dummy' : Dummy variable or variables (in case of a multivariate nb_ts object) used in X12-ARIMA regression; the dummy variables can also include values for forecasts and backcasts if you request them. Either an nb_math_ts object or empty.
- 'dummyType' : Type of dummy. Either 'ao' | {'holiday'} | 'td'
- 'mode' : Seasonal adjustment mode. 'auto' means that series with only positive or only negative numbers will be adjusted in the 'mult' (multiplicative) mode, while series with combined positive and negative numbers in the 'add' (additive) mode. {'auto'} | 'add' | 'logadd' | 'mult' | 'pseudoadd' | 'sign'
- 'maxIter' : Maximum number of iterations for the X12 estimation procedure. Must be numeric. Default is 1500.
- 'maxOrder' : A 1-by-2 vector with maximum order for the regular ARMA model (can be 1, 2, 3, or 4) and maximum order for the seasonal ARMA model (can be 1 or 2). Must be numeric. Default is [1,2].
- 'missing' : Allow for in-sample missing observations, and fill in values predicted by an estimated ARIMA process; if false, the seasonal adjustment will not run and a warning will be thrown. 1 | {0}.
- 'output' : Requested output data, as a string. Either:
 - 'irregular' : For the irregular component,
 - 'seasadj' : For the final seasonally adjusted series. Default.
 - 'seasonal' : For seasonal factors.
 - 'trendcycle' : For the trend-cycle.
 - 'missingvaladj' : For the original series with missing observations replaced with ARIMA estimates.
- 'specFile' : Name of the X12-ARIMA spec file Must be a char. Default is 'default'.
- 'tdays' : Correct for the number of trading days. 1 | {0}
- 'tolerance' : Convergence tolerance for the X12 estimation procedure. Must be numeric. Default is 1e-5.
- 'startDate' : Start date of seasonally adjustment.

```
- 'endDate' : End date of seasonally adjustment.
```

Output:

- obj : An nb_math_ts object with the seasonally/trend adjusted/component of all the stored series of the object.
- x : The original object with the backcast and forecast appended.

Caution : If the object is link to a data source this object will still not be updateable.
- outputfile : A cellstr array with the output files from the seasonal/trend adjustment.
- errorfile : A cellstr array with error files from the seasonal/trend adjustment.
- mdl : A struct with ARIMA model estimates.

Written by Kenneth Sætherhagen Paulsen

■ nb_ts

Go to: [Properties](#) | [Methods](#)

A class representing timeseries data.

Constructor:

```
obj = nb_ts(datasets,NameOfDatasets,startDate,variables)
obj = nb_ts(datasets,NameOfDatasets,startDate,variables,sorted)
```

Input:

- datasets :

Input can be one of the data types listed below:

- xls(x) : A name of the excel spreadsheet to import.
(With or without extension.)

Caution : If you read a specific worksheet, and want the object to be updateable you must provide the extension.

- mat : Name(s) of the .mat files to import. (With or without extension.).

Format of the .mat file:

The .mat file must be saved structure with the fieldnames as the variable names and the fields as the data of the variables, as doubles. Plus

it must have a field with name 'startDate' with the start date of the data, as a string.

- tseries : As an IRIS tseries object.
- dyn_ts : As dyn_ts object(s), each page of the dyn_ts object is added as separate datasets.
- double : As double matrix. Each page of the matrix will be added as separate datasets.
- struct : As a structure of either dyn_ts object or double matrices. Each field is added as separate datasets.
- FAME : A string with the FAME path for where the FAME database you want read is. Must include the extension .db. Not supported by public version.
- SMART : Give 'smart'. Not supported by public version.

You can also give a cell array consisting of one or a combination of the data types mentioned above. Each cell will be added as a new dataset.

Extra: Use the method structure2nb_ts function to add all the fields of the structure as variables in an nb_ts object. (If each field has more page then 1, then each page will be a datasets)

- NameOfDatasets :

Input options will depend on data types listed below:

- xls(x) : A string with the wanted dataset name. Default is the excel file name. If the provided string is a sheet of the read spreadsheet that specific worksheet will be read.
- mat : A string with the wanted dataset name. Default is the .mat file name.
- tseries : A string with the wanted dataset name. Default is 'Dataset1'.
- dyn_ts : A string with the wanted dataset name. Default is 'Dataset1'.
- double : A string with the wanted dataset name. Default is 'Dataset1'.

Caution: When you given double matrices which has more pages then one, either directly or through a struct. Each of the pages will be added as a dataset. The name of this datasets will, if the input NameOfDatasets has the same size as the input datasets, get the name NameOfDatasets{jj}<jj>, where jj is

the page number, or else get the name 'Database<jj>'.

- struct : This input has nothing to say. The dataset names will be given by the fieldnames.
- FAME : A string with the vintage date to fetch. Default is the FAME path name.
- SMART : A string with the context date to fetch. Default is to fetch last context.
- A cell array of the listed types above:

If you give a cell array to the datasets input, this input must be a cell array of strings. If you give a cell array which doesn't have the same size as the cell array given to the datasets input, the datasets which is left gets the name 'Database<jj>', where <jj> stands for this added datasets number.

If you give an empty cell array, i.e. {} all the datasets get the names; 'Database<jj>', where <jj> stands for the added datasets number. Except when the datasets input are given by a cellstr with the excel file names, .mat file names or a FAME path names, then the default dataset names will be set to the same cellstr. Or if some elements of the cell is a struct, then the datasets provided through the struct will get their fieldnames as the dataset names.

- startDate :

The start date of the data. Only needed when you give numerical data as one element of the input datasets (also when numerical data is given through a struct) or when you fetch data from a FAME database (the start date of the fetched data).

Must be a string on the date format given below or an object which is of a subclass of the nb_date class.

Dating convention:

> Yearly	:	'yyyy'.	E.g. '2011'
> Semiannualy	:	'yyyySs'	E.g. '2011S1'
> Quarterly	:	'yyyyQq'.	E.g. '2011Q1'
> Monthly	:	'yyyyMm(m)'.	E.g. '2011M1', '2011M11'
> Weekly	:	'yyyyWw(w)'	E.g. '2011W1'
> Daily	:	'yyyyMm(m)Dd(d)'	E.g. '2011M1D1', '2011M11D11'

- variables :

A cellstr array of the names of the variables.

Needed when the dataset input is:

> Numerical matrix. The number of variables must be of the same as the size of the data (2. dimension).

```

> A struct consisting of double matrices.

> An tseries or an nb_math_ts object. The number of
variables must be of the same as the size of the data
(2. dimension).

> When you want to fetch data from a FAME database. I.e.
the variables fetched.

> When you give a cell array of one of the types mentioned
above.

- sorted : true or false. Default is true.

Output:

- obj : An object of class nb_ts

Examples:

- xls(x):

    One excel spreadsheet

    obj = nb_ts('test1')

    More excel spreadsheets

    obj = nb_ts({'test1','test2',...})

    same as

    obj = nb_ts({'test1','test2',...},{'test1','test2',...})

- mat:

    One .mat file

    obj = nb_ts('test1')

    More .mat files

    obj = nb_ts({'test1','test2',...})

    same as

    obj = nb_ts({'test1','test2',...},{'test1','test2',...})

- nb_math_ts:

    One nb_math_ts object

    obj = nb_ts(test1,'test1','','varName1','varName2',...)

    obj = nb_ts(test1,'','','varName1','varName2',...)

    same as

```

```

obj = nb_ts(test1,'Database1','','...
             {'varName1','varName2',...})

More nb_math_ts objects

obj = nb_ts({test1,test2,...},{'test1','test2',...},...
             '',{'varName1','varName2',...})

obj = nb_ts({test1,test2,...},{},'',...
             {'varName1','varName2',...})

same as

obj = nb_ts({test1,test2,...},...
             {'Database1','Database2',...},'',...
             {'varName1','varName2',...})

- tseries (IRIS:

One tseries object

obj = nb_ts(test1,'test1','','',{'varName1','varName2',...})

obj = nb_ts(test1,'','','',{'varName1','varName2',...})

same as

obj = nb_ts(test1,'Database1','','...
             {'varName1','varName2',...})

More tseries objects

obj = nb_ts({test1,test2,...},{'test1','test2',...},...
             {'varName1','varName2',...})

obj = nb_ts({test1,test2,...},{},'',...
             {'varName1','varName2',...})

same as

obj = nb_ts({test1,test2},{'Database1','Database2',...},...
             '',{'varName1','varName2',...})

- dyn_ts:

One dyn_ts object

obj = nb_ts(test1,'test1')

obj = nb_ts(test1)

same as

obj = nb_ts(test1,'Database1')

More dyn_ts objects

obj = nb_ts({test1,test2,...},{'test1','test2',...})

```

```

obj = nb_ts({test1,test2,...})

same as

obj = nb_ts({test1,test2,...},{'Database1','Database2',...})

-struct:

One struct

> Consisting of dyn_ts object(s):

obj = nb_ts(struct1)

> Consisting of double matrices:

obj = nb_ts(struct1,{},'1994Q1',{'Var1','Var2',...})

More structs

> Consisting of dyn_ts object(s):

obj = nb_ts({struct1,stuct2,...})

> Consisting of double matrices:

obj = nb_ts({struct1,stuct2,...},{},'1994Q1',...
            {'Var1','Var2',...})

- double matrix:

One matrix

obj = nb_ts(double1,'','1994Q1',{'Var1','Var2',...})

obj = nb_ts(double1,'test1','1994Q1',{'Var1','Var2',...})

More matrices

obj = nb_ts({double1,double2,...},{},'1994Q1',...
            {'Var1','Var2',...})

obj = nb_ts({double1,double2,...},{'name1','name2',...},...
            '1994Q1',{'Var1','Var2',...})

One matrix with more pages:

obj = nb_ts(double1,{},'1994Q1',{'Var1','Var2',...})

obj = nb_ts(double1,{'name1','name2',...},'1994Q1',...
            {'Var1','Var2',...})

```

See also:

[nb_date](#), [nb_year](#), [nb_semiAnnual](#), [nb_quarter](#), [nb_month](#), [nb_day](#)
[nb_math_ts](#), [nb_cs](#), [nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- **data**
- **frequency**
- **numberOfObservations**
- **userData**
- **dataNames**
- **localVariables**
- **numberOfVariables**
- **variables**
- **endDate**
- **numberOfDatasets**
- **startDate**

- **data** [↑](#)

The data of the object. As a double or logical.

Inherited from superclass NB_DATASOURCE

- **dataNames** [↑](#)

The names of the different dataset. As a cellstr.

Inherited from superclass NB_DATASOURCE

- **endDate** [↑](#)

The end date of the data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_week, nb_month, nb_quarter, nb_semiAnnual or nb_year.

- **frequency** [↑](#)

Frequency of data:

```
1   : yearly (also used for integers)
2   : semiannually
4   : quarterly
12  : monthly
52  : weekly
365 : daily (Also when dealing with business days,
           which gets nan values for all the skipped days)
```

- **localVariables** [↑](#)

A nb_struct with the local variables of the nb_ts object.
I.e. a field with name 'test' can be reach with the string input %#test. Only for dates and vintage inputs.

Inherited from superclass NB_DATASOURCE

- **numberOfDatasets** [↑](#)

Number of datasets stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **numberOfObservations** [↑](#)

Number of observation of the data stored in the object. As a double.

- **numberOfVariables** [↑](#)

Number of variables stored in the object. As a double.

Inherited from superclass NB_DATASOURCE

- **startDate** ↑

The start date of the data. Given as an object which is of a subclass of the nb_date class. Either: nb_day, nb_week, nb_month, nb_quarter, nb_semiAnnual or nb_year.

- **userData** ↑

Add user data. Can be of any type.

Inherited from superclass NB_DATASOURCE

- **variables** ↑

Variables of the object. As a cellstr.

Inherited from superclass NB_DATASOURCE

Methods:

- [abs](#)
- [acosd](#)
- [acot](#)
- [acoth](#)
- [acsed](#)
- [addDataset](#)
- [addLags](#)
- [addPages](#)
- [addPrefix](#)
- [addVariable](#)
- [agrowth](#)
- [append](#)
- [appendRecursive](#)
- [asCellForMPRReport](#)
- [asecd](#)
- [asin](#)
- [asinh](#)
- [atan](#)
- [atanh](#)
- [avgOAF](#)
- [bkfilter1s](#)
- [breakLink](#)
- [callop](#)
- [callstat](#)
- [checkExpressions](#)
- [conj](#)
- [convertEach](#)
- [acos](#)
- [acosh](#)
- [acotd](#)
- [acsc](#)
- [acsch](#)
- [addDatasets](#)
- [addPageCopies](#)
- [addPostfix](#)
- [addTimeTrend](#)
- [aegrowth](#)
- [and](#)
- [appendFromAnotherPage](#)
- [asCell](#)
- [asec](#)
- [asech](#)
- [asind](#)
- [assignNan](#)
- [atand](#)
- [autocorr](#)
- [bkfilter](#)
- [breakAdj](#)
- [callfun](#)
- [callrtfun](#)
- [ceil](#)
- [cleanRealTime](#)
- [convert](#)
- [corr](#)

- cos
- cosd
- cosh
- cot
- cotd
- coth
- cov
- covidDummy
- createSeasonalDummy
- createShift
- createTimeDummy
- createVarDummy
- createVariable
- csc
- cscd
- csch
- cumnansum
- cumprod
- cumsum
- cutAtDates
- cutAtVintage
- cutSample
- dates
- deleteMethodCalls
- deleteVariables
- demean
- denton
- deptcat
- detrend
- diff
- disp
- doTransformations
- double
- easterDummy
- ecdf
- egrowth
- empty
- epcn
- eq
- estimateDist
- exp
- expand
- expandPeriods
- expm1
- extMethod
- extrapolate
- extrapolateStock
- fillNaN
- floor
- fromRise_ts
- ge
- get
- getClassOfData
- getCode

- getCreatedVariables
- getRealEndDate
- getRealEndDateVariables
- getRealStartDatePages
- getRelease
- getSourceList
- growth
- h2l
- hdi
- histogram
- hpfilter
- iegrowth
- igrowth
- ipcn
- isCrossSectional
- isDouble
- isTimeseries
- isempty
- isnan
- jbTest
- keepPages
- ksdensity
- lag
- le
- log
- log1p
- lt
- getMethodCalls
- getRealEndDatePages
- getRealStartDate
- getRealStartDateVariables
- getSource
- getVariable
- gt
- hasVariable
- head
- horzcat
- hpfilter1s
- iepcn
- interpolate
- isBoolean
- isDistribution
- isRealTime
- isUpdateable
- isfinite
- isscalar
- keepInitial
- keepVariables
- kurtosis
- lastObservation
- lead
- log10
- log2
- map2Distribution

- mavg
- max
- mean
- meanDiff
- meanGrowth
- median
- merge
- merge2Series
- mergeContexts
- min
- minus
- mldivide
- mode
- mpower
- mrdivide
- mstd
- mtimes
- nan
- nan2var
- ne
- not
- objSize
- ones
- or
- packing
- parcorr
- pca
- pcn
- percentiles
- permute
- plot
- plotDist
- plus
- pow2
- power
- q2y
- rand
- rdivide
- reIndex
- real
- realTime2RecCondData
- reallog
- realpow
- realsqrt
- rebalance
- recursiveExtrapolate
- release
- removeObservations
- rename
- renameMore
- reorder
- ret
- rlag
- round

- saveDataBase
- sec
- secd
- sech
- set2nan
- setDayOfWeek
- setInfinite2NaN
- setMethodCalls
- setNewStartDate
- setToNaN
- setValue
- setZero2NaN
- sgrowth
- shrinkSample
- simulate
- sin
- sind
- sinh
- skewness
- sort
- sortProperty
- splitByPublishDates
- splitSample
- splitSeries
- sqrt
- squeeze
- std
- stdise
- stopSorting
- strip
- stripButLast
- stripFcstFromRealTime
- struct
- structure2Dataset
- subAvg
- subSum
- subsasgn
- subsref
- sum
- sumOAF
- summary
- tail
- tan
- tand
- tanh
- times
- timesCS
- toMFfile
- toRise_ts
- toStructure
- toTseries
- todyn_ts
- tonb_cell
- tonb_cs

- `tonb_data`
 - `uminus`
 - `undiff`
 - `unstruct`
 - `update`
 - `uplus`
 - `var`
 - `vertcat`
 - `whiten`
 - `window`
 - `writeTex`
 - `x12Census`
 - `zeros`
-

► **abs** ↑

```
obj = abs(obj)
```

Description:

Take the absolute value of each data elements of the object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **acos** ↑

```
obj = acos(obj)
```

Description:

Take acos of the data stored in the `nb_cs` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acos(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acosd** ↑

obj = acosd(obj)

Description:

acosd(obj) is the inverse cosine, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acosh** ↑

obj = acosh(obj)

Description:

acosh(obj) is the inverse hyperbolic cosine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acosh(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acot** ↑

obj = acot(obj)

Description:

Take acotangent of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acot(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acotd** ↑

obj = acotd(obj)

Description:

acotd(obj) is the inverse cotangent, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acotd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acoth** ↑

obj = acoth(obj)

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acoth(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsc** ↑

obj = acsc(obj)

Description:

Take inverse csc of the data stored in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = acsc(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **acscd** ↑

obj = acscd(obj)

Description:

acscd(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = acscd(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **acsch** ↑

obj = acsch(obj)

Description:

acsch(obj) is the inverse hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = acsch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **addDataset ↑**

```
obj = addDataset(obj,dataset,NameOfDataset,startDate,variables)
```

Description:

Makes it possible to add one dataset to a existing nb_ts object

Caution : If more than one dataset is added to an object the link to the data source is broken!

Input:

- obj : An object of class nb_ts

- dataset : One of the following:

> A string with the name of excel spreadsheet.
(With or whitout extension)

> A string with the name of a .mat file
(With or whitout extension)

> A string with the filepath to a FAME database

> A dyn_ts object

> A tseries object

> A nb_math_ts object

> A numerical matrix.

> A JAVA TS object.

- NameOfDataset : Must be a string with the dataset name. If not given (or given as '') the default name Database<jj> is given. Where jj is the number of added datasets of the object, including the dataset you add with this method.

When fetching from FAME this input could be a vintage tag. I.e. you could specify the vintage you want to fetch from FAME of the given

series. If the vintage tag does not exist it will fetch the last vintage before the given vintage tag.

- startDate : Must be a string or an object which is of a subclass of nb_date with the start date of the data. Have only effect (and then it must be given) when you try to add data from a numerical matrix and fetch data from FAME.
- variables : A cell array of strings with the names of the variables you want to add. Only needed when a numerical matrix of data, a tseries object, a nb_math_ts object or when data from FAME is added to the object.

Must have the same size as the data you add.
(Second dimension)

When fetching data from FAME this input decides which variables to fetch.

Output:

- obj : An nb_ts object with the added dataset.

Examples:

```
obj = nb_ts();
obj = obj.addDataset('excelFileName');
obj = obj.addDataset('excelFileName2','sheetName');
obj = obj.addDataset('matFileName');
obj = obj.addDataset('famePath','','2012Q1',{'QUA_PCPI'});
obj = obj.addDataset(tseriesObject,'','','{'Var1','Var2'});
obj = obj.addDataset(nb_math_tsObject,'','','{'Var1','Var2'});
obj = obj.addDataset([2,2;2,2],'','2011Q1',{'var1','var2'});
```

See also:

[nb_ts](#), [addDatasets](#), [structure2Dataset](#)

Written by Kenneth S. Paulsen

► **addDatasets** ↑

```
obj = addDatasets(obj,datasets,NameOfDatasets,startDate, ...
                   variables)
```

Description:

Makes it possible to add more datasets to a existing nb_ts object.

Input:

Same input as of the constructor but only cell arrays are supported for the inputs 'datasets' and 'NameOfDatasets'. 'NameOfDatasets' could be given as a empty cell; {};

Output:

- obj : An nb_ts object now with the added datasets.

Examples:

See also:

[nb_ts](#), [addDataset](#), [structure2Dataset](#)

Written by Kenneth S. Paulsen

► **addLags ↑**

obj = addLags(obj,nLags)

Description:

Add the wanted number of lags to the object

Input:

- obj : An object of class nb_ts
- nLags : The number of added lags
- type : Either 'lagFast' (default) or 'varFast'

Output:

- obj : An object of class nb_ts

Examples:

obj = addLags(obj,2)

Written by Kenneth S. Paulsen

► **addPageCopies ↑**

obj = addPageCopies(obj,num)

Description:

Add copies of the data of an object and append it as new datasets of the object. Only possible if an object has only one page.

Input:

- obj : An object of class nb_ts
- num : Number of copies to be appended, or a vector of nb_date objects with the end date of each new page.

Output:

- obj : An object of class nb_ts with the added copies of the first dataset in the object.

Examples:

```
data    = nb_ts(rand(20,2),'', '2011Q1', {'Var1','Var2'});  
dataP  = data.addPageCopies(10);  
dates   = vec(nb_quarter('2015Q1'),nb_quarter('2015Q3'));  
dataP2 = data.addPageCopies(dates);
```

Written by Kenneth S. Paulsen

► addPages ↑

```
obj = addPages(obj,DB,varargin)
```

Description:

Add all pages from different nb_ts objects.

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_ts
- DB : An object of class nb_ts
- varargin : Optional number of objects of class nb_ts

Output:

- obj : The object itself with added datasets from the objects given by varargin.

Be aware: If the added datasets does not contain the same variables or dates, the data of the nb_ts object with the tightest window will be expanded automatically to include all the same dates and variables. (the added data will be set as nan)

Examples:

```
obj = obj.addPages(DB);
obj = obj.addPages(DB1,DB2,DB3);
```

Written by Kenneth S. Paulsen

► **addPostfix** ↑

```
obj = addPostfix(obj,postfix)
obj = addPostfix(obj,postfix,vars)
```

Description:

Add a postfix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- postfix : The postfix to add to all the variables of the object. Must be a string
- vars : A cellstr of the variable to add the prefix to. Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with the postfix added to the variables selected

Examples:

```
obj = obj.addPostfix('_NEW');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **addPrefix** ↑

```
obj = addPrefix(obj,prefix)
obj = addPrefix(obj,prefix,vars)
```

Description:

Add a prefix to all the variables in the nb_ts, nb_cs or nb_data object, or a provided variable group.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- prefix : The prefix to add to all the variables of the object.
Must be a string
- vars : A cellstr of the variable to add the prefix to.
Default is all variable of the object.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data with a prefix added to the selected variables

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► addTimeTrend ↑

```
obj = addTimeTrend(obj)
obj = addTimeTrend(obj,start,finish, name)
```

Description:

Add a time trend variable to the nb_ts object

Input:

- obj : A nb_ts object
- start : The start date of the time trend. Default is the start date of the dataset
- finish : The end date of time trend. Default is the end date of the dataset
- name : The name of the added variable. Default is 'timeTrend'

Output:

- obj : A nb_ts object with the added time trend variable

See also:

Written by Kenneth Sætherhagen Paulsen

► **addVariable** ↑

```
obj = addVariable(obj,startDateOfData,Data,Variable)
```

Description:

Add a new variable to a existing nb_ts object (including all pages)

Input:

- obj : An object of class nb_ts
- startDateOfData : The start date of the data of the added variable. Must be a string or a date object. Cannot be outside the window of the dates of the object.
- Data : The data of the added variable. Cannot be outside the window of the data of the object. Must be a double.
- Variable : A string with the added variable name. Cannot be the same as a existing variable. (Then use setValue instead)

Output:

- obj : An object of class nb_ts with the added variable

Examples:

```
obj = addVariable(obj,'2012Q1',[2;2;2;2],'newVariable');
```

Written by Kenneth S. Paulsen

► **aegrowth** ↑

```
obj = aegrowth(obj,percent)
```

Description:

Calculate exact annual growth, using the formula:
 $(x(t) - x(t-lag))/x(t-lag)$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- percent : Either 1 (in percent) or 0 (not in percent). 0 is default.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = aegrowth(obj);  
obj = aegrowth(obj,1);
```

See also:

[agrowth](#), [egrowth](#), [growth](#)

Written by Kenneth S. Paulsen

► **agrowth ↑**

```
obj = agrowth(obj,percent)
```

Description:

Calculate approx annual growth, using the formula:
 $\log(x(t)) - \log(x(t-lag))$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- percent : Either 1 (in percent) or 0 (not in percent). 0 is default.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = agrowth(obj);  
obj = agrowth(obj,1);
```

See also:

[aegrowth](#), [egrowth](#), [growth](#)

Written by Kenneth S. Paulsen

► **and** ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the and operator has evaluated all the data elements of the object.
The data will be a logical matrix

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **append** ↑

```
obj = append(obj,DB,interpolateDate,method,rename,type)
```

Description:

Append a nb_ts object to another nb_ts object.

Caution:

- It is possible to merge datasets with the same variables as long as they represent the same data or have different timespans.
- If the datasets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they the one object can append the other.
- In contrast to the vertcat method, this method will keep the

first inputs end date and cut all the shared dates of the appended object.

- If the frequencies of the objects is not the same the object with the lowest frequency will be converted.

Input:

- obj : An object of class nb_ts
- DB : An object of class nb_ts
- interpolateDate :
 - How to transform the database with the lowest frequency.
 - 'start' : The interpolated data are taken as given at the start of the periods. I.e. use 01.01.2012 and 01.01.2013 when interpolating data with yearly frequency. (Default)
 - 'end' : The interpolated data are taken as given at the end of the periods. I.e. use 31.12.2012 and 31.12.2013 when interpolating data with yearly frequency.
 - method : The method to use when converting:
 - 'linear' : Linear interpolation
 - 'cubic' : Shape-preserving piecewise cubic interpolation
 - 'spline' : Piecewise cubic spline interpolation
 - 'none' : No interpolation. All data in between is given by nans (missing values) (Default)
 - 'fill' : No interpolation. All periods of the higher frequency get the same value as that of the lower frequency.
 - rename :> 'on' : Renames the prefixes (default)
> 'off' : Do not rename the prefixes
 - For more see the 'rename' option of the convert method.
 - type : The type of series to merge.
 - > 'levelGrowth' : Append level data with growth rates. Must be one period growth. The merged series will be in level.
 - > 'growthLevel' : Append growth rates with level data. Must be one period growth. The merged series will be in growth rates.

```
> 'levelLevel' : Append level series, where the
      merging is done in growth rates,
      but the end result is still in
      levels.

> ''          : Standard merging
```

Output:

- obj : An nb_ts object where the datasets from the two nb_ts objects are merged

Examples:

```
obj = append(obj,DB)
obj = obj.append(DB)
```

Written by Kenneth S. Paulsen

► **appendFromAnotherPage** ↑

```
obj = appendFromAnotherPage(obj,page)
```

Description:

Append data from another page where the rest has missing data.

Input:

- obj : A nObs1 x nVar x nPages nb_dataSource object.
- page : The page to append data from.

Output:

- obj : A nObs x nVar x nPages nb_dataSource object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **appendRecursive** ↑

```
obj = appendRecursive(obj,app,dat)
```

Description:

Append data from app by adding new pages to obj.

Caution link to the data source will be broken.

Input:

- obj : A single paged nb_ts object.
- app : An object of class nb_data, nb_cs or double.
- dat : The dates where the data from app should start. Must match the number of pages of app. As a cellstr.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► asCell ↑

```
cellMatrix = asCell(obj,dateFormat,strip)
```

Description:

Return the data of the object as a cell matrix.
(In the dyn_ts format, i.e. looks like a excel spreadsheet.)

Input:

- obj : An object of class nb_ts
- dateFormat : The date format of the dates of the cell.
 - > 'default' :
 - yearly : 'yyyy'
 - semiannually : 'yyyySs'
 - quarterly : 'yyyyQq'
 - monthly : 'yyyyMm(m)'
 - daily : 'yyyyMm(m)Dd(d)'
 - > 'xls' : 'dd.mm.yyyy' for all frequencies
- strip : - 'on' : Strip all observation dates where all the variables has no value.
 - 'off' : Does not strip all observation dates where all the variables has no value. Default.

Output:

- cellMatrix : The nb_ts objects data transformed to a cell.

Examples:

```
cellMatrix = obj.asCell();
cellMatrix = obj.asCell('xls');
cellMatrix = obj.asCell('xls','on');
```

Written by Kenneth S. Paulsen

► **asCellForMPRReport ↑**

```
cellMatrix = asCellForMPRReport(obj,roundoff)
```

Description:

Return the data of the object as a cell matrix, which is rounded off to two decimals and striped

Input:

- obj : An object of class nb_ts
- roundoff : Give the number of wanted decimals. If empty no round-off will be done. Default is 2, i.e. 2.02. 0 decimals is also possible. Must be an integer larger than or equal to 0, or empty (no round-off).

Output:

- cellMatrix : The objects data transformed to a cell.

Examples:

```
cellMatrix = obj.asCellForMPRReport();
cellMatrix = obj.asCellForMPRReport(0);
```

See also:

[asCell](#)

Written by Kenneth S. Paulsen

► **asec ↑**

```
obj = asec(obj)
```

Description:

Take inverse sec of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asecd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sin of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees,
of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = asinh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **assignNan** ↑

```
obj = expand(obj,variables,value)
```

Description:

Assign all the nan observation to the value input.

Input:

- obj : An object of class nb_dataSource.
- variables : A char or a cellstr with the variables to assign nan values.
- value : A scalar number.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj = nb_ts.rand('2012Q1',10,2,3);
obj(1:2,1,:) = nan;
obj = assignNan(obj,'Var1',0)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the inverse tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees, of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = atanh(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **autocorr** ↑

```
obj = autocorr(obj,numLags)
obj = autocorr(obj,numLags,errorBound,alpha)
```

Description:

Compute the sample autocorrelation function (ACF) of all the variables stored in the nb_ts object, and return an nb_data object with the results.

Reference:

[1] Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. Time Series Analysis: Forecasting and Control. 3rd edition. Upper Saddle River, NJ: Prentice-Hall, 1994.

Input:

- obj : An object of class nb_ts.

Caution : This method does not handle nan or infinite for any of the stored data of the nb_ts object!
- numLags : Positive integer indicating the number of lags of the ACF to compute. If empty or missing, the default is to compute the ACF at lags 1. Since ACF is symmetric about lag zero, negative lags are ignored. Must be a scalar.
- errorBound : A string. The data must be stationary. Either:
 - > '' : No errorbounds calculated.
 - > 'asymptotic' : Using asymptotically derived formulas.
 - > 'paramBootstrap' : Calculated by estimating the data generating process. And use the fitted model to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
 - > 'blockbootstrap' : Calculated by blocking the observed series and using these blocks to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
 - > 'copulaBootstrap' : The observed series is bootstrapped based on a copula approach. See Paulsen (2017).
- alpha : Significance value. As a scalar. Default is 0.05.

Optional inputs:

- 'maxAR' : As an integer. See the nb_arima function. Default is 3.
- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- obj : An nb_data object with the autocorrelations stored in the dataset 'Autocorrelations'.
If the errorbounds are to be calculated they are stored in the datasets 'Error bound (lower)' and 'Error bound (upper)'

Examples:

```
obj = autocorr(obj)
obj = autocorr(obj,10)
```

See also:

[nb_ts](#), [nb_data](#), [corr](#), [nb_autocorr](#)

Written by Kenneth Sæterhagen Paulsen

► **avgOAF** ↑

```
obj = avgOAF(obj,sumFreq)
```

Description:

Take the average over a lower frequency. Ignoring nan values!

I.e. if frequency is quartely, this function can take the average over all the quarters of a year. And it will return the average over the year in all quarters of the that year

Input:

- obj : An object of class nb_ts
- sumFreq : The frequency to average over

Output:

- obj : An object of class nb_math_ts where the data is averaged over a lower frequency

Examples:

```
obj = nb_ts.ones('2012Q1',8,1);
obj = obj.avgOAF(1)
```

Written by Kenneth S. Paulsen

► **bkfilter** ↑

```
obj = bkfilter(obj,low,high)
```

Description:

Do band pass filtering of all the dataseries of the object.
(Returns the gap). Will strip nan values when calculating the filter

Input:

- obj : An object of class nb_ts
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_ts with the band pass filtered data.

Examples:

```
obj = bkfilter(obj,8,32);
obj = obj.bkfilter(8,32);
```

See also:

[bkfilter1s](#)

Written by Kenneth S. Paulsen

► **bkfilter1s** ↑

```
obj = bkfilter1s(obj,low,high)
```

Description:

Do one sided band pass-filtering of all the dataseries of the object. (Returns the gap) Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_ts
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_ts with the one-sided band pass filtered data.

Examples:

```
obj = obj.bkfilterls(12,24);
obj = bkfilterls(obj,12,24);
```

See also:

[bkfilter](#)

Written by Kenneth S. Paulsen

► **breakAdj** ↑

```
obj = breakAdj(obj,method,varargin)
```

Description:

The intent of this method is to set observations around break points to nan, and interpolate the observation at these break-points. I.e. if you have a level series that has a break, and want smooth growth rates. In this case you can calculate the growth rates of the level series, and then use this method to smooth out the growth rate at the break-points. Then you can use the inverse growth methods to transform back to a level series without breaks.

Input:

- obj : An object of class nb_ts
- method : The wanted interpolation method to use. See the nb_interpolate function.

Optional input:

- A set of nb_date objects or one line chars with the dates that should be set to nan and interpolated.

Output:

- obj : An object of class nb_ts.

See also:

[nb_interpolate](#), [growth](#), [egrowth](#), [pcn](#), [epcn](#), [igrowth](#), [iegrowth](#), [ipcn](#)
[iepcn](#)

Written by Kenneth S. Paulsen

► **breakLink** ↑

obj = breakLink(obj)

Description:

Break link to source

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **callfun** ↑

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb_dataSource object(s). The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb_dataSource.getClassOfData to find the class of the data of the object.

Input:

- obj : An object of class nbDataSource.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nbDataSource objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nbDataSource object. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d      = nb_ts.rand('2012Q1',10,3);
d1     = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2     = callfun(d,d,'func','plus') % Same as d2 = d + d !
[s1,s2] = callfun(d,'func',@size) % Same as [s1,s2] = size(d)
```

See also:

[nb_ts.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► callop ↑

```
obj = callop(obj,another,func)
obj = callop(obj,another,func,type)
```

Description:

Call mathematical operator on the data of the object. In contrast to calling the operator itself on the object, this method does not try to match the same variables from the two inputs, but instead operate in the same way as mathematical operators do on double matrices (bsxfun). The time domain is matched though!

Caution: The following operators are supported:
> @plus or 'plus'

```
> @minus or 'minus'  
> @times or 'times'  
> @rdivide or 'rdivide'  
> @power or 'power'
```

Input:

- obj : An object of class nb_dataSource.
- another : An object of class nb_dataSource.
- func : One of the above listed functions.
- type : Either:
 - > 'keep' : Keep the variable names of the object with most variables. If they have the same number of variables, they are taken from the first object.
 - > 'rename' : The new variable names represent the operation that has been done to the separate series. So if you divide an object with one series named 'Var1' with another object with one series named 'Var2', the new name will be 'Var1./Var2'. (Default)

Output:

- varargout : All output from the provided function which results in either a double, logical or nb_distribution with the same size as obj input will be converted to a nb_dataSource method. The rest will be given as return by the func function when applied to the data of the object.

Examples:

```
d1 = nb_ts(rand(10,1),'', '2012Q1', {'Var1'});  
d2 = nb_ts(rand(10,1),'', '2011Q1', {'Var2'});  
d3 = callop(d1,d2,@minus);  
  
d1 = nb_ts(rand(10,1),'', '2012Q1', {'Var1'});  
d2 = nb_ts(rand(10,2),'', '2011Q1', {'Var2','Var3'});  
d3 = callop(d1,d2,@minus);
```

See also:

[nb_ts.getClassOfData](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► [callrtfun ↑](#)

```
out = callrtfun(obj,another,'func',func)
out = callrtfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the data of the nb_ts object(s) representing real-time data. The data of the object is normally of class double, but may also be of class logical or nb_distribution. Use nb_dataSource.getClassOfData to find the class of the data of the object.

Caution: For the nb_ts object ot be taken as a real-time datasets it must have the context dates stored in the dataNames property on the format 'yyyymmdd' and only represent one variable.

Input:

- obj : An object of class nb_ts.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function @(x)x will be used.
- 'name' : Name of the new variable. Default is 'new'.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class. nb_ts objects are converted to a matrix with elements of class double, logical or nb_distribution.

Output:

- out : An object of class nb_ts.

See also:

[nb_dataSource.getClassOfData](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **callstat** ↑

```
obj = callstat(obj,func)
obj = callstat(obj,func,varargin)
```

Description:

Call a in-built or user defined function on the data of the nb_dataSource object(s) that goes from the data of the object have more than one variable to only having one. E.g. when taking the mean over a set of variables.

Input:

- obj : An object of class nb_dataSource.
- func : A function handle (or one line char) that maps a nObs x nVar x nPage double to nObs x 1 x nPage double, or that maps a nObs x nVar x nPage double to nObs x nVar x 1 double.

Optional input:

- 'name' : Set the name of the new variable, as a one line char. Default is to use the str2func on the provided function handle.
- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- obj : An object of class nb_dataSource with only one variable.

Examples:

```
d = nb_ts.rand('2012Q1',10,3);
d1 = callstat(d,@(x)mean(x,2),'name','mean')
```

See also:

[nb_ts.mean](#), [nb_ts.std](#), [nb_ts.var](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► ceil ↑

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = ceil(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **checkExpressions** ↑

```
obj = checkExpressions(obj,expression)
obj = checkExpressions(obj,expression,shift)
```

Description:

Check expressions, and add to dataset

Input:

- obj : An object of class nb_ts

- expression : A cell on the format:

```
{Name1, expression1, description1;...
  Name2, expression2, description2};
```

The first column sets the name of the created variable.

The second column sets the expression to apply to the existing variables in the object. All methods of the nb_math_ts class that pserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: expression2 can depend on the Name1 variable.

Caution: If Name1 is a name of an existing variable in the object, it will be replaced.

- shift : An object of class nb_ts storing the shift/trend variables. These are added to the matching variable before the transformations are applied. If empty, this input is ignored. Default is [].

Output:

- obj : An object of class nb_ts

Written by Kenneth SÃ¶terhagen Paulsen

► **cleanRealTime** ↑

```
obj = cleanRealTime(obj)
```

Description:

Secure that the real time data has new observations for each period. You have to options 'delete' or 'fill'.

Input:

- obj : An nb_ts object returned by the nb_fetchRealTimeFromFame function.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► **conj** ↑

```
obj = conj(obj)
```

Description:

conj(obj) is the complex conjugate of the elements of obj.
For a complex element x of obj, conj(x) = REAL(x) - i*IMAG(x).

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = conj(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **convert** ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_ts object

Input:

- obj : An object of class nb_ts
- freq : The new frequency of the data. As an integer:
 - > 1 : yearly
 - > 2 : semi annually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- method : The method to use. Either:
 - From a high frequency to a low:
 - > 'average' : Takes averages over the subperiods
 - > 'diffAverage' : Uses a weighted average, so that the diff operators can be converted correctly. E.g. when converting from monthly to quarterly, and the level of the variable of interest is aggregated to quarterly data using averages, but you want to convert monthly data on diff to quarterly. The formula used is:
$$\text{diff_Y}(q) = \frac{1}{3}\text{diff_Y}(m) + \frac{2}{3}\text{diff_Y}(m-1) + \text{diff_Y}(m-2) + \frac{2}{3}\text{diff_Y}(m-3) + \frac{1}{3}\text{diff_Y}(m-4)$$
 - > 'diffSum' : Same as 'diffAverage', but now the aggregation in levels are done using a sum instead.
 - > 'discrete' : Takes last observation in each subperiod (Default)
 - > 'first' : Takes first observation in each subperiod
 - > 'max' : Takes maximal value over the subperiods
 - > 'min' : Takes minimum value over the subperiods
 - > 'sum' : Takes sums over the subperiods
 - From a low frequency to a high:
 - > 'linear' : Linear interpolation
 - > 'cubic' : Shape-preserving piecewise cubic interpolation
 - > 'spline' : Piecewise cubic spline interpolation
 - > 'none' : No interpolation. All data in between is given by nans (missing values) (Default)
 - > 'fill' : No interpolation. All periods of the higher frequency get the same value as that of the lower frequency.
 - > 'daverage' : Uses the Denton 1971 method using that the average of the intra low frequency periods should preserve the average.
 - > 'dsum' : Uses the Denton 1971 method using that the sum of the intra low frequency

periods should preserve the sum.

Caution: This input must be given if you provide some of the optional inputs.

Caution: All these option will by default use the first period of the subperiods as base for the conversion. Provide the optional input 'interpolateDate' if you want to set the base to the end periods instead ('end').

Optional input:

- 'includeLast' : Give this string as input if you want to include the last period, even if it is not finished. Only when converting to lower frequencies

E.g: obj = obj.convert(1,'average',...
'includeLast');

- 'rename' : Changes the postfixes of the variables names.

E.g. If you covert from the frequency 12 to 4 the prefixes of the variable names changes from 'MUA' to 'QUA', if the prefix exist.

E.g: obj = obj.convert('yearly','average',...
'rename');

Optional input (...,'inputName', inputValue,...):

- 'fill' : Either 'nan' (default) or 'zeros'. This set how to fill the values that are not set when method is set to 'none'.

- 'interpolateDate' : The input after this input must either be 'start', 'end' or a scalar integer.

Date of interpolation. Where 'start' means to interpolate the start date of the periods of the old frequency, while 'end' uses the end date of the periods. If a scalar integer is added it is taken as 'end' minus the given number of periods.

Default is 'start'.

Caution : Only when converting to higher frequency.

Caution : For weekly data, the 'dayOfWeek' options is added. See nb_ts.setDayOfWeek for more on this.

E.g: obj = obj.convert(365,'linear',...
'interpolateDate','end');

Output:

- obj : An nb_ts object converted to wanted frequency.

Examples:

```
obj = obj.convert(4,'average');
obj = obj.convert(365,'linear','interpolateDate','end');
obj = obj.convert(1,'average','includeLast');
obj = obj.convert(1,'average','rename');
```

See also:

[nb_ts.convertEach](#)

Written by Kenneth S. Paulsen

► convertEach ↑

```
obj = convertEach(obj,freq,method,varargin)
```

Description:

Convert the frequency of each series of the nb_ts object.

Caution: If you want to apply same conversion method to all series use nb_ts.convert instead.

Input:

- obj : An object of class nb_ts.
- freq : The new frequency of the data. As an integer:
 - > 1 : yearly
 - > 2 : semi annually
 - > 4 : quarterly
 - > 12 : monthly
 - > 52 : weekly
 - > 365 : daily
- methods : The methods to use to convert each series. As a 1 x numberOfVariables cell array. Set to '' to use default.
See the method input to the nb_ts.convert to get information on the supported method you can use to convert the series.

Optional input:

- Same as for the nb_ts.convert method. Will be applied to all series.

Output:

- obj : An nb_ts object converted to wanted frequency.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

► **corr ↑**

obj = corr(obj,varargin)

Description:

Calculate the correlation of the timeseries of the nb_ts object

Caution: Calculates the correlation matrix of the variables of the nb_ts object, if no optional inputs are given. If 'lags' is given this function calculates the correlation with the number of wanted lags for each variable with itself. (And only with itself)

Input:

- obj : An object of class nb_ts

Optional input ('propertyName',PropertyValue):

- 'lags' : An integer with the number of lags you want to calculate the correlation for. (Here for each variable)
- 'robust' : true or false. Give true if you want to remove missing observations during calculations. (Also inf)
- 'type' : 'pairwise' or 'default'. 'pairwise' calculate the correlation matrix for the selected lag or lead, while 'default' calculates the correlation with itself for lags from 1 to lags and the same for lead.

Output:

- obj : An nb_cs object with the wanted correlation

Examples:

```
corrMatrix = corr(obj);
corrMatrix = corr(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► **cos** ↑

```
obj = cos(obj)
```

Description:

Take cos of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = cos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

cosd(obj) is the cosine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cosd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cosh** ↑

```
obj = csch(obj)
```

Description:

`cosh(obj)` is the hyperbolic cosine of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cot** ↑

```
obj = cot(obj)
```

Description:

Take cotangent of the data stored in the `nb_ts`, `nb_cs` or `nb_data` object

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
obj = cot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_DATASOURCE`

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

`cotd(obj)` is the cotangent of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = cotd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **coth** ↑

```
obj = coth(obj)
```

Description:

`coth(obj)` is the hyperbolic cotangent of the elements of `obj`.

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = coth(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **cov** ↑

```
obj = cov(obj,varargin)
```

Description:

Calculate the covariances of the timeseries of the nb_ts object

Caution: Calculates the covariance matrix of the variables of the nb_ts object, if no optional inputs are given. If 'lags' is given this function calculates the covariance with the number of wanted lags for each variable with itself. (And only with itself)

Input:

- obj : An object of class nb_ts

Optional input ('propertyName',PropertyValue):

- 'lags' : An integer with the number of lags you want to calculate the covariance for. (Here for each variable)
- 'robust' : true or false. Give true if you want to remove missing observations during calculations. (Also inf)
- 'type' : 'pairwise' or 'default'. 'pairwise' calculate the covariance matrix fro the selected lag or lead, while 'default' calculates the covariance with itself for lags from 1 to lags and the same for lead.

Output:

- obj : An nb_cs object with the wanted covariances

Examples:

```
corrMatrix = cov(obj);  
corrMatrix = cov(obj,'lags',2);
```

Written by Kenneth S. Paulsen

► **covidDummy** ↑

```
obj = covidDummy(obj)  
obj = covidDummy(obj,covidDummyDates)
```

Description:

A dummy for the covid-19 episode.

Input:

- obj : An object of class nb_ts.
- covidDummyDates : A cellstr with the dates of the covid-19 crisis.
Default is the period 16.03.2020 – 31.12.2021.
Hint: Use nb_month(3,2020):nb_month(12,2021).

Output:

- obj : An object of class nb_ts.

Examples:

```
Generate random data:  
data = nb_ts.rand('2018Q1',12,3)  
data = covidDummy(data)  
dataM = nb_ts.rand('2018M1',36,3)  
dataM = covidDummy(dataM)  
dataW = nb_ts.rand('2020W1',40,3)  
dataW = covidDummy(dataW)
```

See also:

[nb_covidDummyNames](#), [nb_covidDates](#)

Written by Kenneth Sæterhagen Paulsen

► **createSeasonalDummy** ↑

obj = createSeasonalDummy(obj,type)

Description:

Creates and adds seasonal dummy variables to the nb_ts object.

Input:

- obj : An object of class nb_ts.
- type : Centered/ uncentered

Output:

- obj : An object of class nb_ts. (With the added dummy variable)

Examples:

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **createShift ↑**

```
[obj,shift]      = createShift(obj,variables,expression,output)
[obj,shift,plotter] = createShift(obj,variables,expression,output)
```

Description:

Remove trend/shift from variables and add them in a new database shift.

Each element of the expression input specify one of the following:

- How to demean the data or detrend data.
- How to get the trend given a calculate the gap. See the FGTS option below.

Functionalities:

- {'constant',value} or {'constant',value} :

With this option you can make the series stationary by subtracting a constant over the whole timespan. This will also be the projection going forward. (Could also be a mathematical expression)

Optional inputs: 'cmavg'. These inputs work the same as is the case for the 'hpfilter'.

- {'avg'} or {'avg',...} :

Subtract the mean from the dataseries. Including short term forecast. This will also be the projection going forward.

Optional inputs: 'wfcst', 'whist' and 'cmavg'. These inputs work the same as is the case for the 'hpfilter'.

- {'exponentialSmoothen',weight} or {'exponentialSmoothen',weight,...}:

Exponential smoother. See the exptrend function for more.

Optional inputs:

- Same as for the 'hpfilter' case.
- {'hpfilter',lambda} or {'hpfilter',lambda,...}:

HP filter with lambda given by lambda (a number). Switch 'hpfilter' to 'hpfilterls' to do one-sided HP-filter.

Optional inputs:

- 'wfcst'

When 'wfcst' is given, this function includes all the data after the date given in the cell after this one to compute the hpfilter. See last option for more on de-trending sub-periods.

- 'whist'

When 'whist' is given, this function includes all the data before the date given in the cell before this one to compute the hpfilter. See last option for more on de-trending sub-periods.

Both 'wfcst' and 'whist' does nothing if the hpfilter is not combined with other functions in a cell!

- 'randomWalk' : Use a 20 periods projection of the level in the series by a random walk before filtering the series.
- 'randomwalkgrowth' : Use a 20 periods projection of the growth in the series by a random walk before filtering the series.
- 'cmavg' : Apply the central moving average to the gap, i.e. nb_mavg(gap,1,1).
- function_handle : A function_handle that takes the time-series of gap as it's only input.
E.g: @(x)nb_mavg(x,1,1)

- {'bkfilter',low,high} or {'bkfilter',low,high,...} :

Band pass filter removing all the freq. outside the frequencies given by low and high. Switch 'bkfilter' to 'bkfilterls' to do one-sided band pass filter.

Optional inputs:

- Same as for the 'hpfilter' case.
- {'linear'} or {'linear',...} :
Apply the linear filter to the data.

Optional inputs:

- Same as for the 'hpfilter' case.
- {'M2T','ar',trend,arCoeff} or
{'M2T','ar',trend,arCoeff,trendStart,lambda}
(M2T = Merge To Trend)

If you don't want a constant trend in the future you can use this option. (Instead of {'end'}). This function will close

the gap between the given growth rate in the trend, at the date given before this sub-period, and the growth rate given by trend, with a AR process. Where the AR coefficient is given by arCoeff. And it will return the level trend implied by this growth "path".

- trend : Must be a number with the long term growth rate in the trend.
- arCoeff : Must be a scalar number with the ar coefficient
- trendStart : Must be a number with the starting value of the growth rate in the trend. Can also be set to 'hpfilter'. In this case it starts out with the growth rate of a hp-filter trend with lambda set by the lambda input. Optional.
- lambda : A scalar double with the lambda used for the hp-filter if trendStart is set to 'hpfilter'. Default is 3000. Optional.
- {'M2T','int',trend}

This function does the same as the one above, but it uses a linear growth "path" instead of the one implied by the AR process to close the "growth gap".

- {'FGTS',expression} (FGTS = From Gap To Shift)

If you have the gap but need the shift variable you can use this function. Where expression must be a string with the expression to evalute. I.e. the expression of data input which the gap was cumputed of. E.g. 'log(Var1)'.

- A cell where you give different ways to detrend data for different sub-periods. The cell must be given in the following way:

```
{{'constant',2}, '2011Q2', {'int',2,3}, '2012Q2', {'end'}}
```

Where each odd cell element is representing the period up and including the date given as the even cell elements which follows. The cell given above will therefore subtract 2 up and including 2011Q2, and will subtract the line starting at 2 in 2011Q3 and ending at 3 in 2012Q2. From then on it will subtract the last value given at 2012Q2 for all periods ahead.

Options for each cell (up and including the date given as the next cell):

- {'constant',value} : as above
- {'avg'} : as above
- {'hpfilter',lambda,...} : as above
- {'bkfilter',low,high,...} : as above
- {'M2T','ar',trend,arCoeff} : as above

- {'FGTS','expression'} : as above

Options for the last sub-period (last cell element):

- {'end'} : Project the trend using a random walk.
- {'endgrowth'} : Project the trend using a random walk in the growth rate.
- {'M2T','ar',trend,arCoeff} : as above

Input:

- obj : An object of class nb_ts
- nSteps : Number of steps to extrapolate the shift variable
- variables : The variables to be removed a trend/shift. As a cell array or a string.
Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the trend/shift of all the variables.
As a cell array with the same size as the 'variables' input.
See the description of this method for the supported de-trending methods.
- output : If given as 'shift' the first output will be the shift output instead.

Output:

- obj : A nb_ts object where the listed variables has been detrended.
- shift : A nb_ts object storing the trend/shift variables. All the variables not included in the variables input will be given a shift variable of zero.
- plotter : A nb_graph_ts object that plots the de-trending of the data. Use the graphInfoStruct method or the nb_graphInfoStructGUI class to produce the graphs.

Caution: If dealing with real-time data, only the last vintage is displayed.

Examples:

```
obj = obj.createShift({'var1','var2',...},...
    {{'avg'},'2012Q1',{'end'}},{{'constant',2}},...);
```

Written by Kenneth S. Paulsen

► **createTimeDummy** ↑

```
obj = createTimeDummy(obj, nameOfDummy, date, condition)
```

Description:

Creates and adds a dummy variable to the nb_ts object. Will test the conditions given the date provided.

Input:

- obj : An object of class nb_ts.
- nameOfDummy : Name of the added dummy variable.
- date : The date to test. Either a string, a nb_date object or a 1 x 2 cell array, where each element is either a string, a nb_date object.
- condition : The conditions to test.
 - '<' : True before. Not supposed if date is a cell array.
 - '>' : True after. Not supposed if date is a cell array.
 - '<=' : True before and including. Not supposed if date is a cell array.
 - '>=' : False before and including. Not supposed if date is a cell array.
 - '==' : True for the given period (default).
 - '~=': True for all but the given period.

Output:

- obj : An object of class nb_ts. (With the added dummy variable)

Examples:

```
obj = nb_ts(ones(5,2),'', '2012', {'Var1','Var2'});  
obj = createTimeDummy(obj,'Dummy','2013','>');  
obj = createTimeDummy(obj,'Dummy2',{ '2013','2015'},'==');  
obj = createTimeDummy(obj,'Dummy3',{ '2013','2015'},'~=');
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **createVarDummy** ↑

```
obj = createVarDummy(obj, nameOfDummy, varargin)
```

Description:

Creates and adds a dummy variable to the nb_ts object. Will test the conditions given for the variable provided by the testedVariable input.

Input:

- obj : An object of class nb_ts.
- nameOfDummy : Name of the added dummy variable.
- varargin : The conditions to test.
 - '<' : Less than. Second input must be a double (scalar), an nb_ts object (if more variables it must be less than all the stored variables) or a string with the variable to test against. The third input must either be '|' or '&' indicating if you want to check that this condition and ('&') or or ('|') the next is satisfied. The and operator(s) will have presedens over the or operator.
 - '>' : Greater than. (Or else same as '<')
 - '<=' : Less than or equal to. (Or else same as '<')
 - '>=' : Less than or equal to. (Or else same as '<')
 - '==' : Equal to. (Or else same as '<')
 - '~=': Not equal to. (Or else same as '<')

Output:

- obj : An object of class nb_ts. (With the added dummy variable)

Examples:

```
obj = nb_ts([1,2;-3,1;-1,3],'', '2012', {'Var1','Var2'});  
obj = createVarDummy(obj,'Dummy','Var1','>',0,'&');  
obj = createVarDummy(obj,'Dummy','Var1','>','Var2','&');  
obj = createVarDummy(obj,'D2','Var1','>',0,'|','Var2','<',1,'&');  
obj = createVarDummy(obj,'D2','Var1','>',0,'&','Var2','<',1,'&');
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **createVariable** ↑

```
obj = createVariable(obj, nameOfNewVariable, expression)
obj = createVariable(obj, nameOfNewVariable, expression, parameters, varargin)
```

Description:

Create variable(s) and add them to the nb_ts object dataset(s)

- This function only support methods of the class nb_math_ts for the expressions. Type the following in the command window;

```
methods(nb_math_ts)
```

Input:

- obj : An object of class nb_ts
- nameOfNewVariable : The created variable names as a cell array or a string.
Must have the same size as the 'expressions' input.
- expression : The expressions of how to create the data of all the created variables.
As a cell array or a string.
Must have the same size as the 'nameOfNewVariables' input.
- parameters : A struct with the parameters that can be used in the expressions. Caution: They will be added as series, so you need to use elementwise operators (.*, ./, .^)!

Optional input:

- 'overwrite' : true or false. Set to true to allow for overwriting of variables already in the data of the object. Default is false.
- 'warning' : true or false. Set to true to give warning if expressions fail (instead of error). Will result in series having all nan value, when warning is thrown.

Output:

- obj : An nb_ts object with the created variable(s) added.

Examples:

```
obj = obj.createVariable('var3','var1./var2');  
  
obj = obj.createVariable({'var3','var4',...},...  
{'var1./var2','var2./var1',...});
```

See also:

[nb_math_ts](#)

Written by Kenneth S. Paulsen

► **csc** ↑

```
obj = csc(obj)
```

Description:

Take csc of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = cscd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **csch** ↑

```
obj = csch(obj)
```

Description:

csch(obj) is the hyperbolic cosecant of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = csch(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **cumnansum** ↑

```
obj = cumnansum(obj)
```

Description:

Cumulative sum of the series of the object. Ignoring nan values.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumnansum(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumprod** ↑

```
obj = cumprod(obj, dim)
```

Description:

Cumulativ product of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- obj : A nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative product of the old objects 'data' property.

Examples:

```
obj = cumprod(obj);
obj = cumprod(obj, 2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulativ sum of the series of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : In which dimension the cumulativ sum should be calculated. Default is the first dimension.

Output:

- obj : An nb_ts, nb_cs or nb_data object where the 'data' property of the object is now the cumulative sum of the old objects 'data' property.

Examples:

```
obj = cumsum(obj);  
obj = cumsum(obj,2);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **cutAtDates** ↑

```
obj = cutAtDates(obj,endDates)
```

Description:

Cut the time-series of the variables of the object.

Input:

- obj : An object of class nb_ts
- endDates : A nPages x nVars nb_date array.

Output:

- obj : An object of class nb_ts

Written by Kenneth S. Paulsen

► **cutAtVintage** ↑

```
obj = cutAtVintage(obj, lags)
```

Description:

Cut each vintage that include forecast at the period of the vintage tag minus the number of wanted lags.

Input:

- obj : An object of class nb_ts
- lags : The number of lags to do to find the end date at each vintage.
Default is 1.

Output:

- obj : An object of class nb_ts

Examples:

```
obj = cutAtVintage(obj);
```

Written by Kenneth S. Paulsen

► **cutSample ↑**

```
obj = cutSample(obj)
```

Description:

Remove leading and trailing nan values, will give an error if the sample still includes nan values.

Input:

- obj : An object of class nb_ts
- tolerate : Tolerate number of trailing nan. An integer greater than or equal to 0.

Output:

- obj : An object of class nb_ts

Examples:

```
out = cutSample(in);
```

See also
nb_ts.shrinkSample

Written by Kenneth S. Paulsen

► **dates** ↑

```
allDates = dates(obj)
allDates = dates(obj,format)
```

Description:

Get all the dates of the object. As a cellstr.

Input:

- obj : An object of class nb_ts
- format : See the method toDates in the nb_date class for supported formats

Output:

- allDates : A cellstr array of the dates of the object

Examples:

```
allDates = obj.dates();
allDates = obj.dates('xls');
```

Written by Kenneth S. Paulsen

► **deleteMethodCalls** ↑

```
obj = deleteMethodCalls(obj,c)
```

Description:

Delete some method calls of the object. The object needs to be updatable.

Caution: To set method calls use setMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- sourceNr : The source index to delete the methods from
- methodNrs : A index with the methods to delete.

Output:

- obj : An object of class nb_ts, nb_data or nb_cs

See also:

[setMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **deleteVariables ↑**

obj = deleteVariables(obj,deletedVar)

Description:

Delete variables from the current nb_ts, nb_data or nb_cs object (letter size has something to say)

If some of the variables you specify in deletedVar does not exist in the obj, this will not have anything to say for the obj. (The variables cannot be deleted, because the variables do not exist.)

Input:

- obj : An object of class nb_ts, nb_data or nb_cs

- deletedVar : Must be a char or a cellstr

Delete all the variable names, given in the char array or cellstr array deletedVar, of current the object.

NB! If deletedVar is just one string and include a * as the last letter, this function will delete all the variable names, from the object, that start with the letters which is in front of the * in the string.

i.e. obj = deleteVariables(obj,'DPQ*');

Deletes all the variables in the obj which has variable names starting with 'DPQ'.

If deletedVar is just one string and include a * as the first letter, this function will delete all the variable names, from the object, that include the letters which follows in the string.

i.e. obj = deleteVariables(obj, '*shift');

Deletes all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : A nb_ts, nb_data or nb_cs object with the variables specified in deletedVar deleted.

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = deleteVariables(obj, {'Var1','Var2'});  
obj = deleteVariables(obj, char('Var1','Var2'));  
obj = deleteVariables(obj, obj.variables);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **demean** ↑

```
obj = demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- obj : A nb_dataSource object.
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_dataSource object.

Written by Tobias Ingebrigtsen

Inherited from superclass NB_DATASOURCE

► **denton** ↑

```
obj = denton(obj,z,k,type,d)
```

Description:

The Denton method of transforming a series from low to high frequency.

See Denton (1971), Adjustment of Monthly or Quarterly Series to Annual Totals: An Approach Based on Quadratic Minimization.

Input:

- obj : A nobs x nvars x npage nb_ts object with the main variables to transform.
- z : A nobs*k x nvars x npage double with observations on the judgment.
- k : The number intra low frequency observations. If you have annual data and want out quarterly data use 4.
- type : Either 'sum', 'average', 'first' or 'last'. 'average' is default.
- d : 1 : first differences, 2 : second differences.

Output:

- x : Output series as a nobs*k x nvars x npage nb_ts object.

See also:

[nb_ts.convert](#), [nb_denton](#)

Written by Kenneth Sæterhagen Paulsen

► **deptcat ↑**

obj = deptcat(obj,varargin)

Description:

Depth concatenation (add pages of different objects) (No short notation)

Be aware: If the added datasets does not contain the same variables or dates, the data of the nb_ts object with the tightest window will be expanded automatically to include all the same dates and variables. (the added data will be set as nan)

Input:

- obj : An object of class nb_ts
- varargin : Optional number of nb_ts objects

Output:

- obj : An object of class nb_ts where all the objects datasets are added into.

Examples:

```
obj = obj.deptcat(DB);
obj = obj.deptcat(DB,DB2,DB3);
```

See also:

[addPages](#)

Written by Kenneth S. Paulsen

► **detrend** ↑

```
obj          = detrend(obj,method)
obj          = detrend(obj,method,output,varargin)
[obj,trendObj,plotter] = detrend(obj,method,output,varargin)
```

Description:

Detrending nb_ts object.

Input:

- obj : As a nb_ts object.
- method : Either {'linear'}, 'linearls', 'mean', 'hpfilter', 'hpfilterls', 'bkfilter', 'bkfilterls', 'baiPerron' or 'exponentialsmoother'.
- output : If given as 'trend' the first output will be the trend output instead. 'Normal' is default.
- varargin : Additional inputs. When:
 - > 'exponentialsmoother' : One more input must be given. I.e. weight on previous value. See exptrend function for more.
 - > 'hpfilter' or 'hpfilterls' : One more input must be given. I.e. lambda.
 - > 'bkfilter' or 'bkfilterls' : Two more inputs must be given. I.e. lowFreq and highFreq.
 - > 'baiPerron' : Two input may be given:
 - The first:
 - * 'bic' : Bayesian information criterion is used to select the breaks. Default.
 - * 'lwz' : The Liu, Wu and Zidek information criterion is used to select the breaks.
 - * integer : An integer with the number of breaks to identify.
 - The second:
 - * integer : The maximum number of breaks to allow when using a information criterion to choose the number of breaks. Default is 2.

Output:

- obj : As a nb_ts object with the detrended data.
- trendObj : As a nb_ts object with the trend.
- plotter : A nb_graph_ts object. Use the graphInfoStruct method or the nb_graphInfoStructGUI class on the returned object.

Examples:

```
obj = nb_ts.rand('2012Q1',10,3);
obj = detrend(obj,'linear');
obj = detrend(obj,'hpfilter','','1600');
obj = detrend(obj,'bkfilter','','6,32');
```

See also:

[createShift](#)

Written by Kenneth Sæterhagen Paulsen

► diff ↑

```
obj = diff(obj)
obj = diff(obj,lags)
obj = diff(obj,lags,skipNaN)
```

Description:

Calculate diff, using the formula: $x(t) - x(t-lag)$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- lags : The number of lags in the diff formula, default is 1.
- skipNaN : -1 : Skip nan while using the diff operator. (E.g. when dealing with working days.)
 - 0 : Do not skip nan values. Default.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = diff(obj);  
obj = diff(obj,4);
```

See also:

[nb_ts](#)

Written by Kenneth S. Paulsen

► **disp ↑**

```
disp(obj)
```

Description:

Display object (In the command window)

Input:

- obj : An object of class nb_ts

Output:

The dataset displayed in the command window.

Examples:

```
obj (Note without semicolon)
```

Written by Kenneth S. Paulsen

► **doTransformations ↑**

```
[obj,shift,plotter] = doTransformations(obj,expressions,fcstHorizon,varargin)
```

Description:

Do transformations on the variables of the object.

Input:

- obj : A nb_ts object

- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While

the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the nb_ts.createVariable method.

> third column : Any expression that can be interpreted by the nb_ts.createShift method.

> fourth column : Comments

- fcstHorizon : The forecast horizon. As an integer. Default is 0.

Optional input:

- 'warning' : true or false. Set to true to give warning if expressions fail (instead of error). Will result in series having all nan value, when warning is thrown.

Output:

- obj : A nb_ts object storing the de-trended created variables.
- shift : A nb_ts object storing the shifts/trend of the created variables.
- plotter : A nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,     shift,      description
    'VAR1_G',   'pcn(VAR1)', 'avg',    'VAR1 growth'
    'VAR2_G',   'pcn(VAR2)', 'avg',    'VAR2 growth'
    'VAR3_G',   'pcn(VAR3)', 'avg',    'VAR3 growth'};

obj = obj.doTransformations(expressions)
```

See also:

[nb_ts.createVariable](#), [nb_ts.createShift](#)

Written by Kenneth Sæterhagen Paulsen

► **double ↑**

```
dataOfObject = double(obj)
```

Description:

Get the data of the object as a double matrix

Caution : If the data of the object is represents as an object of class nb_distribution the mean is returned.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- dataOfObject : The data of the nb_ts, nb_cs or nb_data object

Examples:

```
dataOfObject = double(obj)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **easterDummy** ↑

```
obj = easterDummy(obj,greedy)
```

Description:

A seasonal dummy for easter. It utilizes the eastermethod to estimate when easter occurs every year.

Input:

- obj : An object of class nb_ts.
- greedy : Logical. If true this method will only return a dummy for the month when Easter Sunday occurs, otherwise it will return a dummy for the months where the easter week occurs (i.e. may overlap different months). Default is true.

Output:

- obj : An object of class nb_ts.

Examples:

```
Generate random data:  
data = nb_ts.rand('2012M1',10,3)  
data1 = easterDummy(data,1)  
data2 = easterDummy(data,0)
```

See also:

► **ecdf** ↑

```
obj = ecdf(obj,varargin)
```

Description:

A method for creating an nb_ts object with the Empirical (Kaplan-Meier) cumulative distribution function. It utilizes the inbuilt ecdf function to estimate the distribution. See ecdf for more information on the optional inputs for the function.

Input:

- obj : An nb_ts object with the data.

Optional input:

- 'recursive' : True or false. If true, the CDF will be estimated recursively. Default is false.
- 'recStart' : Sets the start date for the recursive estimation. Default is the start date of the obj + 9 observations. As a string or as an object of class nb_date.
- 'recEnd' : Sets the end date for the recursive estimation. Default is the end date of the object. As a string or as an object of class nb_date.
- 'estStart' : Sets the start of the estimation sample. As a string or as an object of class nb_date.

Output:

- obj : A numIter x numVar nb_ts object where each observation point contains an nb_distribution object with the estimated parameters of the distribution.

Examples:

```
f = nb_ts(randn(1000,1),'', '2010M1',{'v1'});  
obj = f.ecdf('recStart','2012M1','recEnd','2012M12', 'recursive',1);
```

See also:

[nb_ts.ksdensity](#)

► **egrowth** ↑

```
obj = egrowth(obj)
obj = egrowth(obj,lag,stripNaN)
```

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- lag : The number of lags in the growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = egrowth(obj);
obj = egrowth(obj,4);
```

See also:

[growth](#), [aegrowth](#), [agrowth](#)

Written by Kenneth S. Paulsen

► **empty** ↑

```
obj = empty(obj)
```

Description:

Empty the existing nb_ts object.

Input:

- obj : An object of class nb_ts

Output:

- obj : An empty nb_ts object

Examples:

```
obj = empty(obj)
```

Written by Kenneth S. Paulsen

► **epcn ↑**

```
obj = epcn(obj);
obj = epcn(obj,lag,stripNaN)
```

Description:

Calculate exact percentage growth, using the formula:
100*(x(t) - x(t-1))/x(t-1) of all the timeseries of the nb_ts
object.

Input:

- obj : An object of class nb_ts
- lag : The number of lags in the growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = epcn(obj); % period-on-period growth
obj = epcn(obj,4); % 4-periods growth
```

Written by Kenneth S. Paulsen

► **eq ↑**

```
a = eq(a,b)
```

Description:

Test if the one object is equal (==) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **estimateDist ↑**

```
dist = estimateDist(obj,type)
```

Description:

Estimate distributions of the series in the nb_ts object.

Caution : This method strip all nan values.

Input:

- obj : An object of class nb_ts.
- type : The type of distribution to estimate. See the 'type' property of the nb_distribution class for the supported types. (Some may not be possible to estimate using 'mle' and/or 'mme'.) Default is 'normal';

Caution : If estimator is equal to 'kernel', this input is not used.

```

Caution : If type is set to 'hist', no estimator is
          used but the returned nb_distribution
          object instead represent a histogram.

- estimator : Either 'mle' (maximum likelihood), 'mme' (methods
               of moments) or 'kernel' (normal kernel density
               estimation). Default is 'mle'.

- dim       : The dimension to estimate densities over. Either
               1, 2 or 3. Default is 1.

- output    : Either 'nb_distribution' (default) or 'nb_dataSource'.
               If 'nb_dataSource' is chosen the output will be a nb_ts
               object if dim equal 2 or 3, while it will be a nb_cs object
               if dim equal 1.

```

Optional input:

- varargin : Optional input given to the nb_ksdensity function. Please see help for that function.

Output:

- dist : An object of class nb_distribution, nb_ts or nb_cs.

See also:

[nb_distribution](#)

Written by Kenneth Sæterhagen Paulsen

► **exp ↑**

`obj = exp(obj)`

Description:

Take exp of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = exp(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE
```

► **expand** ↑

```
obj = expand(obj,newStartDate,newEndDate,type,warningOff)
```

Description:

Expand the current nb_ts object with more dates and data

Input:

- obj : An object of class nb_ts
- newStartDate : The wanted new start date of the data.
- newEndDate : The wanted new end date of the data.
- type : Type of the appended data :
 - 'nan' : Expanded data is all nan (default)
 - 'zeros' : Expanded data is all zeros
 - 'ones' : Expanded data is all ones
 - 'rand' : Expanded data is all random numbers
 - 'obs' : Expand the data with first observation (before) or last observation after
- warningOff : Give 'off' to suppress warning if no expansion has taken place. Both 'newStartDate' and 'newEndDate' is inside the window.

Output:

- obj : An nb_ts object with expanded timespan

Examples:

```
obj = expand(obj,'1900Q1','2050Q1');
obj = expand(obj,'1900Q1','2050Q1','zeros');
```

Written by Kenneth S. Paulsen

► **expandPeriods** ↑

```
obj = expandPeriods(obj,periods,type)
```

Description:

Expand the data of the object by the number of wanted periods.

Input:

- obj : An object of class nb_ts.
- periods : The number of periods to expand by. If negative the added periods will be added before the start date of the object.
- type : Type of the appended data :
 - 'nan' : Expanded data is all nan (default)
 - 'zeros' : Expanded data is all zeros
 - 'ones' : Expanded data is all ones
 - 'rand' : Expanded data is all random numbers
 - 'obs' : Expand the data with first observation (before) or last observation after

Output:

- obj : An object of class nb_ts.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

► expml ↑

```
obj = expml(obj)
```

Description:

`exp(obj)` is the exponential of the elements of `obj`, `e` to the `obj`.
`expml(obj)` computes `exp(obj)-1`, compensating for the roundoff in
`exp(obj)`.
(For small real `X`, `expml(X)` should be approximately `X`, whereas the
computed value of `EXP(X)-1` can be zero or have high relative error.)

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the
data are equal to $e.^obj.data-1$

Examples:

```
obj = expm1(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **extMethod ↑**

```
obj = extMethod(obj,method,variables,postfix,varargin)
```

Description:

This method can call a nb_dataSource method on a selected variables, and merge the result with the old object by adding a postfix to the new variables.

This method is the same as calling:

```
obj1 = window(obj,'','','variables);
method = str2func(method);
obj     = method(obj,varargin{:});
obj1   = addPostfix(obj1,method);
obj     = merge(obj,obj1);
```

But without replicating the links property unnecessary many times!

Caution : If the object is not updateable this will result in the same as the code above!

Input:

- obj : An object of class nb_ts, nb_data or nb_cs
- method : A str with the method to call. Must be a method of the nb_ts, nb_data or nb_cs class that does not change other dimensions then the second dimension!
- variables : A cellstr with the variables of the object to call the method on. Must be included in the object.
- postfix : A string with the postfix. If empty, the old variables will be replaced by the new ones.

Optional inputs:

These will be the inputs casted to the method except the object itself.

Extra for nb_ts and nb_data:

- '<start>' : The start date/obs of the function evaluation. Given as the second input to the window method call.
- '<end>' : The end date/obs of the function evaluation. Given as the third input to the window method call.

Output:

- obj : A nb_ts, nb_data or nb_cs object

Examples:

```
obj = nb_ts.rand(1,10,3);
obj = extMethod(obj,'lag',{'Var1'},'lag1',1)
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **extrapolate** ↑

```
obj = extrapolate(obj,variables,toDate)
obj = extrapolate(obj,variables,toDate,varargin)
```

Description:

Extrapolate last of observation of one variable to the wanted date.

Caution: This method only works on a single paged nb_ts object.

Input:

- obj : An object of class nb_ts
- variables : A string with the name of the variable to extrapolate, or a cellstr of the variables to extrapolate. Default is to extrapolate all variables.
Caution: When 'method' is set to copula only the selected variables are included in the calculation of the extrapolated values.
- toDate : Extrapolate to the given date. If the date is before the end date of the given variable no changes will be made, and with no warning. Can also be a integer with the number of periods to extrapolate.

Optional inputs:

- 'method' :
 - > 'end' : Extrapolate series by using the last observation of each series. I.e. a random walk forecast. Default.
 - > 'copula' : This approach tries to estimate the unconditional marginal distribution of each series in the dataset and the (auto)correlation structure that describe the connection between these series. Some series may have more observations than others!

Caution : This method works when the unconditional distribution from which the series are assumed to be drawn from are the same over time. E.g. the series must have the same volatility over the period. The second assumption is that all the series are stationary. Use the 'check' input to test for non-stationarity by a ADF test for unit-root. If it is found to have a unit root the series will be differenced when forecasted, and then transformed back to level after that step.

- > 'ar' : Extrapolate each series individually using a fitted AR model.
- > 'var' : Extrapolate all series using a fitted VAR model. Max number of variables is set to 10 for this option.
- 'alpha' : Significance level of ADF test for stationary series. Only an option for the 'copula' method.
- 'check' : Check for stationary series. If not found to be stationary the series will be differenced and extrapolated with this transformation before transformed back to its old level. Only an option for the 'copula' method.
- 'nLags' : Number of lags to include when calculating the correlation matrix used by the copula. I.e. the number of historical observations to condition on when forming the conditional correlation matrix.
- 'draws' : The number of draws used to estimate the mean, when using the 'copula' method.
- 'constant' : Give true to include constant in the AR models used to extrapolate the individual series. Default is false.

Output:

- obj : An nb_ts object with extrapolate series.

Examples:

```
obj = nb_ts.rand('1900',50,2,1);
obj = extrapolate(obj,'Var1','1951');
```

Written by Kenneth S. Paulsen

► **extrapolateStock ↑**

```
obj = extrapolateStock(obj,stock,flow,depreciation)
```

Description:

Extrapolate stock with the flow from where the stock ends.

```
stock(t+1) = (1 - depreciation)*stock(t) + flow(t)
```

Input:

- obj : An object of class nb_dataSource.
- stock : A one line char with the name of the stock series.
- flow : A one line char with the name of the flow series.
- depreciation : A scalar number between 0 and 1.

Output:

- obj : An object of class nb_dataSource.

Written by Kenneth Sæterhagen Paulsen

► **fillNaN ↑**

```
obj = fillNaN(obj)
obj = fillNaN(obj,date,variables)
```

Description:

Fill in for nan values with last valid observation.

Input:

- obj : An object of class nb_ts.
- date : The end date of the returned dataset. Either a nb_date object or string with the date, or a number indicating how many periods back in time from the date today, i.e. 0 is today, -1 is past period and 1 is next period. Default is to use the end date of the object.
Caution : If the given date is before the end date of the data of the object no extrapolation is done.
- variables : A cellstr with the variables to fill in nan for. Default is all variables.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► **floor** ↑

```
obj = floor(obj)
```

Description:

`floor(obj)` rounds the data elements of `obj` to the nearest integers towards minus infinity

Input:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Output:

- `obj` : An object of class `nb_ts`, `nb_cs` or `nb_data`

Examples:

```
out = floor(in);
```

Written by Andreas Haga Raavand

Inherited from superclass `NB_DATASOURCE`

► **fromRise_ts** ↑

```
data_ts = nb_ts.fromRise_ts(obj)
```

Description:

Transform from an RISE ts object or a struct of rise ts objects to an `nb_ts` object.

Input:

- `obj` : An object of class `ts`, or a structure of `ts` objects.

Output:

- `data_ts` : An `nb_ts` object.

Written by Kenneth S. Paulsen

► **ge** ↑

```
a = ge(a,b)
```

Description:

Test if the one object is greater than or equal to (\geq) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **get ↑**

```
propertyValue = get(obj,propertyName)
```

Description:

Get the properties of the object

Input:

- obj : An object of class nb_ts
- propertyName : Name of the property you want to get:
 - 'data' : The data of the object as a double
 - 'dataNames' : Name of the object datasets, as cellstr array
 - 'endDate' : The end date of the object as nb_date

```

- 'frequency'           : The frequency of the data, as
                           double
- 'numberOfDatasets'   : Number of datasets (pages) of the
                           object. Size of the third
                           dimension
- 'numberOfObservations' : Number of observations of the
                           object. Size of the first
                           dimension
- 'numberOfVariables'  : Number of variables of the object.
                           Size of the second dimension
- 'startDate'          : The start date of the object as
                           nb_date
- 'variables'          : The variables of the object, given
                           as a cellstr array

- 'links'               : A structure of the data source
                           links.

- 'sorted'              : true or false

- 'updateable'          : 1 if updateable, 0 otherwise.

```

Output:

```
propertyValue : The property value of the object for the wanted
                  property
```

Examples:

```
dataAsDouble = get(obj,'data');
variables    = get(obj,'variables')
```

Written by Kenneth S. Paulsen

► **getClassOfData ↑**

```
cl = getClassOfData(obj)
```

Description:

Get the class of the data stored by the object.

Input:

- obj : An object of class nb_dataSource.

Output:

- cl : Name of the class that the data of the object is stored as.

See also:

nb_ts.isDistribution, nb_ts.isDouble
nb_ts.isBoolean

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getCode** ↑

```
code = getCode(obj,preamble)
code = getCode(obj,preamble,'inputName', inputValue,...)
```

Description:

Get the LaTeX code of a table representing the data of the object

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_ts
- preamble : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
- 'combine' : If the nb_ts object consist of two or three pages (datasets) this option can be used to combine all the data in one table. Either 1 or 0. Default is 0.
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'dateformat' : Must be a string. 'default' is default.
 - > 'default'
 - > 'fame'
 - > 'xls'
 - > 'NBNorsk' ('NBNorwegian')
 - > 'NBEnglish' ('NBEngelsk')
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:
 - 'Huge', 'huge', 'LARGE', 'Large', 'large', 'normalsize' (default), 'small',

```

'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.

- 'justification' : Sets the justification of the caption of the
table. Must be a string.

Either 'justified','centering' (default),
'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.

- 'language'      : Either 'english' (default) or 'norwegian'
('norsk').

- 'lookUpMatrix'  :

Sets how the given mnemonics (variable names) should map to
different languages. Must be cell array on the form:

{'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};

Or a .m file name, as a string. The .m file must include
(and only include) what follows:

lookUpMatrix = {

'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};

Be aware : Each of the given description can be multi-lined
char, which will result in multi-line text of the
table.

- 'orientation'   : The orientation of the table. Either:

    > 'horizontal' : The dates as columns
    > 'vertical'   : The types as columns

- 'precision'     : Sets the precision of the numbers in the
table. Must be a string. Default is '%3.2f'.
I.e. two decimals.

- 'rotation'      : The rotation of the table in LaTeX. Either:

    > 'sidewaystable' : A sideways table
    > 'landscape'     : Rotate the whole page
    > ''              : No rotation

- 'rowColor1'      : Color of every second row of the table
starting from the second row. Same options
as was the case for the 'firstRowColor'
option.

- 'rowColor2'      : Color of every second row of the table
starting from the third row. Same options
as was the case for the 'firstRowColor'
option.

```

- 'rowColor3' : Color of every third row of the table starting from the fourth row. Only an option in combination with the 'combine' input set to 1 and that the nb_ts object has 3 pages. Same options as was the case for the 'firstRowColor' option.
- 'NaNrepl' : Replaces NaN's in a table by chosen symbol. Give as string, can be empty. Default: 'NaN', i.e. no replacement.

Output:

- code : A char with the tex code of the table

Examples:

```
obj = nb_ts(rand(10,3),'', '2012Q1', {'Var1','Var2','Var3'});
code = obj.getCode(1)
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► [getCreatedVariables ↑](#)

```
created = getCreatedVariables(obj)
```

Description:

Get a N x 2 table of the variables created using the createVariable method. The first column list the created variables, while the second column list the expressions.

Input:

- obj : An object that is a subclass of nb_dataSource.

Output:

- created : A N x 2 cell array. See description of this method for more.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► [getMethodCalls ↑](#)

```
[s,c] = getMethodCalls(obj)
```

Description:

Get all method calls as a cell matrix. The object needs to be updatable to get a list of methods called on the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.
- c : A cell matrix with a table of the method called on the object and its input. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getRealEndDate** ↑

```
realEndDate = getRealEndDate(obj)
realEndDate = getRealEndDate(obj,format,type)
realEndDate = getRealEndDate(obj,format,type,variables)
```

Description:

Get the real end date of the nb_ts object. I.e the last observation which is not nan or infinite.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.

Output:

- realEndDate : The last observation of the object which is not nan or infinite. As a string

Examples:

```
realEndDate = obj.getRealEndDate();  
realEndDate = obj.getRealEndDate('pprnorsk');
```

Written by Kenneth S. Paulsen

► getRealEndDatePages ↑

```
realEndDate = getRealEndDatePages(obj)  
realEndDate = getRealEndDatePages(obj,format,type)  
realEndDate = getRealEndDatePages(obj,format,type,variable,pages)
```

Description:

Get the real end date of each page of the nb_ts object. I.e the last observation which is not nan or infinite for each page.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.
- pages : A double array with whole numbers with the pages to check.

Output:

- realEndDate : The last observation of the object which is not nan or infinite for each page. Either a cellstr or a vector of objects which is of a subclass of the nb_date class.

Examples:

```
realEndDate = obj.getRealEndDatePages();  
realEndDate = obj.getRealEndDatePages('pprnorsk');
```

Written by Kenneth S. Paulsen

► **getRealEndDateVariables ↑**

```
realEndDate = getRealEndDateVariables(obj)  
realEndDate = getRealEndDateVariables(obj,format)  
realEndDate = getRealEndDateVariables(obj,format,type,page)
```

Description:

Get the real end date of each variable of the nb_ts object. I.e the last observation which is not nan or infinite for each variable.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- pages : A double array with whole numbers with the pages to check.

Output:

- realEndDates : The last observation of the object which is not nan or infinite for each variable. Either a cellstr or a vecor of objects which is of a subclass of the nb_date class.

Examples:

```
realEndDates = obj.getRealEndDateVariables();  
realEndDates = obj.getRealEndDateVariables('nb_date');
```

Written by Kenneth S. Paulsen

► **getRealStartDate** ↑

```
realStartDate = getRealStartDate(obj)
realStartDate = getRealStartDate(obj,format,type,variables)
```

Description:

Get the real start date of the nb_ts object. I.e the first observation which is not nan or infinite.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
> 'xls'
> 'pprnorsk' or 'mprnorwegian'
> 'pprengelsk' or 'mprenglish'
> 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
> 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.

Output:

- realStartDate : The first observation of the object which is not nan or infinite. As a string

Examples:

```
realStartDate = obj.getRealStartDate();
realStartDate = obj.getRealStartDate('pprnorsk');
```

Written by Kenneth S. Paulsen

► **getRealStartDatePages** ↑

```
realEndDate = getRealStartDatePages(obj)
realEndDate = getRealStartDatePages(obj,format,type)
realEndDate = getRealStartDatePages(obj,format,type,variable,pages)
```

Description:

Get the real end date of each page of the nb_ts object. I.e the last observation which is not nan or infinite for each page.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- variables : A cellstr of the variables to check.
- pages : A double array with whole numbers with the pages to check.

Output:

- realEndDate : The first observation of the object which is not nan or infinite for each page. Either a cellstr or a vector of objects which is of a subclass of the nb_date class.

Examples:

```
realEndDate = obj.getRealStartDatePages();  
realEndDate = obj.getRealStartDatePages('pprnorsk');
```

Written by Kenneth S. Paulsen

► **getRealStartDateVariables ↑**

```
realEndDate = getRealStartDateVariables(obj)  
realEndDate = getRealStartDateVariables(obj,format)  
realEndDate = getRealStartDateVariables(obj,format,type,pages)
```

Description:

Get the real start date of each variable of the nb_ts object. I.e the first observation which is not nan or infinite for each variable.

Input:

- obj : An object of class nb_ts
- format : The date format returned.
 - Return a string:
 - > 'xls'
 - > 'pprnorsk' or 'mprnorwegian'
 - > 'pprengelsk' or 'mprenglish'
 - > 'default' (otherwise)
 - Return a date object which is of a subclass of the nb_date class:
 - > 'nb_date'
- type : Either 'any' (default) or 'all'.
- pages : A double array with whole numbers with the pages to check.

Output:

- realStartDates : The first observation of the object which is not nan or infinite for each variable. Either a cellstr or a vector of objects which is of a subclass of the nb_date class.

Examples:

```
realEndDates = obj.getRealStartDateVariables();
realEndDates = obj.getRealStartDateVariables('nb_date');
```

Written by Kenneth S. Paulsen

► **getRelease ↑**

```
obj = release(obj,num)
```

Description:

Get the wanted release from the real-time data of the object.

This method is not restricted to only updatable real-time data as is the case with nb_ts.release. Instead the dataNames property must be given with the 'yyyymmdd' format.

Caution : This method will break the link to the data source of updateable objects!

Input:

- obj : An object of class nb_ts. The vintages dates stored in the dataNames property on the format 'yyyymmdd'.

Caution: The number of variables stored is limited to one!

- num : The release number. 1 for the first release, and so on. 0 for final vintage.

Output:

- obj : An object of class nb_ts with only one page.

See also:

[nb_ts.release](#)

Written by Kenneth Sæterhagen Paulsen

► **getSource** ↑

```
sourceType = getSource(obj)
sourceType = getSource(obj,type)
```

Description:

Get source type.

Input:

- obj : An object of class nb_dataSource.
- type : Give 'program' to get the general source. Default is to give the specific type of source.

Output:

- sourceType : A one line char with the source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getSourceList** ↑

```
s = getSourceList(obj)
```

Description:

Get the source list of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- s : List of sources, as a cellstr. Will return {} if the object is not updatable.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **getVariable ↑**

```
variableData = getVariable(obj,variableName,startDate,...  
                           endDate, pages)
```

Description:

Get the data of a variable, as a double

Input:

- obj : An object of class nb_ts
- variableName : The variable you want the data of. Must be given as a string (name of variable) or a cellstr.
- startDate : Start date of the return data of the given variable. As a string or an object which is of a subclass of nb_date
- endDate : End date of the return data of the given variable. As a string or an object which is of a subclass of nb_date
- pages : Pages of the return data of the given variable. As a double or logical array, e.e. [1:2] or [1,3,5].

Output:

- variableData : The data of the variable you have given. Return [] if not found.

Examples:

```
variableData = getVariable(obj,'Var1');  
variableData = getVariable(obj,'Var1','2012Q1');  
variableData = getVariable(obj,'Var1','2012Q1','2012Q4');  
variableData = getVariable(obj,'Var1','2012Q1','2012Q4',[1:3]);
```

Written by Kenneth S. Paulsen

► **growth** ↑

```
obj = growth(obj)
obj = growth(obj,lag,stripNaN)
```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- lag : The number of lags in the approx. growth formula, default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = growth(obj);
obj = growth(obj,4);
```

See also:

[egrowth](#), [aegrowth](#), [agrowth](#)

Written by Kenneth S. Paulsen

► **gt** ↑

```
a = gt(a,b)
```

Description:

Test if the one object is greater then ($>$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a > b;  
obj = 2 > b;  
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► h21 ↑

```
obj = h21(obj,freq,type)
```

Description:

Will for each observation calculate the weighted cumulative sum over the last N periods (Including the present). The weights will depend on the frequency and the selected type. Examples will be given in the case of monthly data and freq = 4;

- 'levelSummed' : $Y(q) = Y(m) + Y(m-1) + Y(m-2)$
- 'diffSummed' : $Y(q) = Y(m) + 2*Y(m-1) + 3*Y(m-2) + 2*Y(m-3) + Y(m-4)$
- 'levelAverage' : $Y(q) = 1/3*(Y(m) + Y(m-1) + Y(m-2))$
- 'diffAverage' : $Y(q) = 1/3*Y(m) + 2/3*Y(m-1) + Y(m-2) + 2/3*Y(m-3) + 1/3*Y(m-4)$

Input:

- obj : An object of class nb_ts.
- freq : The low frequency to convert high frequency observations to.
- type : See description of method. Default is 'levelAverage'

Output:

- obj : An object of class nb_ts. Caution: The frequency of the nb_ts will not be converted. See nb_ts.convert in this case.

See also:

[nb_ts.convert](#)

► **hasVariable** ↑

```
[ret,location] = hasVariable(obj,variable)
```

Description:

Test if an nb_cs object has a given variable

Input:

- obj : An object of class nb_cs
- variable : The variable name as a string. Can also be a cellstr with more variables.

Output:

- ret : 1 if found otherwise 0
- location : The location of the variable if found. Otherwise [].

Examples:

```
ret          = obj.hasVariable('Var1')
[ret,location] = hasVariable(obj,'Var1')
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **hdi** ↑

```
obj = hdi(obj,perc)
```

Description:

Calculate the highest density interval (hdi) over the third dimension of the data of the object.

Input:

- obj : An object of class nb_ts
- limits : A double with the wanted mass of the hdi of the data. E.g. 0.9, or [0.10,0.20].

Output:

- obj : A nb_ts object with the limits of the hdis stored as different pages of the object. See the dataNames property for the ordering.

Be aware that the hdis may not be overlapping! This means that the number of returned limits may be higher than the number of specified limits*2. So in the case of a bimodal distribution a the pages lb2_* and ub2_* represents the second area of the given limit.

Written by Kenneth Sæterhagen Paulsen

► head ↑

```
head(obj,nRows,page)
```

Description:

Display the first n rows of a nb_ts object.

Input:

- obj : An nb_ts object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_ts](#), [window](#), [tail](#)

Written by Per Bjarne Bye

► histogram ↑

```
data          = histogram(obj)
[data,plotter] = histogram(obj,M)
```

Description:

Create a histogram of a object containing only one variable and one page!

Input:

- obj : A nb_dataSource object with size N x 1 x 1.
- M : The number of bins of the histogram.

Output:

- data : A nb_cs object with size M x 1 x 1.
- plotter : A nb_graph_cs object with the histogram plotted.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► horzcat ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation of nb_ts objects. ([a,b]) This is only possible if the different objects contain different variables or have variables with the same values (start and/or end dates can differ). Uses the merge method.

If the start and/or end dates differ, nan values will be appended.

Input:

- a : An object of class nb_ts
- varargin : Optional number of nb_ts objects

Output:

- obj : An nb_ts object with all the variables from the different nb_ts object merge into one dataset

Examples:

```
obj = [a,b];  
obj = [a,b,c];
```

See also:

[merge](#)

Written by Kenneth S. Paulsen

► hpfilter ↑

```
obj = hpfilter(obj,lambda)  
obj = hpfilter(obj,lambda,perc,fcstLen)
```

Description:

Do hp-filtering of all the dataseries of the nb_ts object.
(Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_ts
- lambda : The lambda of the hp-filter
- perc : Set to true to calculate the gap as (gap/trend)*100.
- fcstLen : The forecast length. Extrapolates the original series using the average of the last 4 periods. Default is 0.

Output:

- obj : An nb_ts object with the hp-filtered timeseries.

Examples:

```
gap = hpfilter(data,3000);
trend = data-gap;
```

See also:

[hpfilter](#)

Written by Kenneth S. Paulsen

► **hpfilterls** ↑

```
obj = hpfilterls(obj,lambda)
obj = hpfilterls(obj,lambda,perc,fcstLen)
```

Description:

Do one-sided hp-filtering of all the dataseries of the nb_ts object. (Returns the gap). Will strip nan values when calculating the filter.

Input:

- obj : An object of class nb_ts
- lambda : The lambda of the hp-filter
- perc : Set to true to calculate the gap as (gap/trend)*100.
- fcstLen : The forecast length. Extrapolates the original series using the average of the last 4 periods. Default is 0.

Output:

- obj : An nb_ts object with the hp-filtered timeseries.

Examples:

```
gap = hpfilterls(data,3000);
trend = data-gap;
```

See also:

[hpfilter](#)

Written by Kenneth S. Paulsen

► iegrowth ↑

```
obj = iegrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact growth, i.e. the inverse method of the egrowth method of the nb_ts class

Input:

- obj : An object of class nb_ts
- InitialValues : A nb_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_ts object with the indicies.

Examples:

```
obj = iegrowth(obj);
obj = iegrowth(obj,100);
obj = iegrowth(obj,[78,89]);

See also
nb_ts.egrowth, nb_ts.igrowth, nb_ts.growth
```

Written by Kenneth S. Paulsen

► **iepcn** ↑

```
obj = iepcn(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact percentage growth, i.e. the inverse method of the epcn method of the nb_ts class

Input:

- obj : An object of class nb_ts
- InitialValues : A nb_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_ts object with the indicies.

Examples:

```
obj = iepcn(obj);
obj = iepcn(obj,100);
obj = iepcn(obj,[78,89]);
```

See also:

[nb_ts.epcn](#)

Written by Kenneth S. Paulsen

► **igrowth** ↑

```
obj = igrowth(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries wich represent the series growth. Inverse of log approx. growth, i.e. the inverse method of the growth method of the nb_ts class

Input:

- obj : An object of class nb_ts
- InitialValues : A nb_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken growth over.

Output:

- Output : An nb_ts object with the indicies.

Examples:

```
obj = igrowth(obj);  
obj = igrowth(obj,100);  
obj = igrowth(obj,[78,89]);
```

See also
nb_ts.growth, nb_ts.egrowth, nb_ts.iegrowth

Written by Kenneth S. Paulsen

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate the data of the nb_data object. Will discard leading and trailing nan values.

Input:

- data : An nb_ts, nb_cs or nb_data object.
- method :
 - 'nearest' : Nearest neighbor interpolation
 - 'linear' : Linear interpolation. Default
 - 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - 'pchip' : Shape-preserving piecewise cubic interpolation
 - 'cubic' : Same as 'pchip'
 - 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : An nb_ts, nb_cs or nb_data object with the interpolated data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ipcn ↑**

```
obj = ipcn(obj,initialValues)
obj = ipcn(obj,initialValues,periods,startAtNaN)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method of the nb_ts class

Input:

- obj : An object of class nb_ts
- InitialValues : A nb_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.

- periods : The number of periods the initial series

has been taken growth over.

- startAtNaN : true or false. If true it will start taking the inverse growth when the nb_ts object with initialValues are nan, instead using the first not nan observation in the same input. Default is false.

Output:

- Output : An nb_ts object with the indicies.

Examples:

```
obj = ipcN(obj);
obj = ipcN(obj,100);
obj = ipcN(obj,[78,89]);
```

See also:

[nb_ts.pcn](#)

Written by Kenneth S. Paulsen

► **isBoolean** ↑

```
ret = isBoolean(obj)
```

Description:

Check if the data of the nb_dataSource object is of class logical (boolean, i.e. true or false).

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isCrossSectional** ↑

```
ret = isCrossSectional(obj)
```

Description:

Test if this object is a cross sectional data object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_cs, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isDistribution** ↑

```
ret = isDistribution(obj)
```

Description:

Check if the data of the nb_dataSource object is of class nb_distribution

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isDouble** ↑

```
ret = isDouble(obj)
```

Description:

Check if the data of the nb_dataSource object is of class double (numeric)

Input:

- obj : An object of class nb_dataSource.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **isRealTime** ↑

ret = isRealTime(obj)

Description:

Is the dataset representing linked real-time data?

Input:

- obj : An object of class nb_ts.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

► **isTimeseries** ↑

ret = isTimeseries(obj)

Description:

Test if a nb_dataSource object is a time series object.

Input:

- obj : An object of class nb_dataSource

Output:

- ret : true if object is of class nb_ts, otherwise false.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isUpdateable** ↑

```
ret = isUpdateable(obj)
```

Description:

Test if this object is updateable.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : 1 if updateable, otherwise 0.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is empty. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Test if each element of a nb_ts, nb_cs or nb_data object is finite.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is finite, otherwise 0.

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Test if each element of an nb_ts, nb_cs or nb_data object is nan.

Caution : Non-updateable operation.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : The object with data as logical. 1 if an element is nan, otherwise 0.

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **isscalar** ↑

```
ret = isscalar(obj)
```

Description:

Test if a nb_ts, nb_cs or nb_data object is a scalar. Will always return 0, as an object of class nb_ts, nb_cs or nb_data is not a scalar.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- ret : false

Examples:

```
0 = isscalar(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **jbTest ↑**

```
[test,pval] = jbTest(obj)
```

Description:

Jargue-Bera test for normality. Null hypothesis of the test is that the series y is normal.

Input:

- obj : An object of class nb_ts.

- dim : The dimension to test for normality. Either 1, 2 or 3. Default is 1.

- out : If given as 'pval' the pval output is returned as the first output, otherwise the order is the standard way.

Output:

- test : The test statistics:
 > dim == 1: An object of class nb_cs with size 1 x nVars x nPage.
 > dim == 2: An object of class nb_ts with size nObs x 1 x nPage.
 > dim == 3: An object of class nb_ts with size nObs x nVars.

- pval : The test p-value.
 > dim == 1: An object of class nb_cs with size 1 x nVars x

```
nPage.  
> dim == 2: An object of class nb_ts with size nObs x 1 x nPage.  
> dim == 3: An object of class nb_ts with size nObs x nVars.
```

Written by Kenneth SÃ¶terhagen Paulsen

► **keepInitial** ↑

```
obj = keepInitial(obj, seasonalPattern, start)
```

Description:

Keep initial value for the variables in the object.

Input:

- obj : An object of class nb_ts.
- seasonalPattern : Give true to keep the seasonal pattern. Default is false.
- start : From the date to start.

Output:

- obj : An object of class nb_ts.

Written by Kenneth SÃ¶terhagen Paulsen

► **keepPages** ↑

```
obj = keepPages(obj, pages)
```

Description:

Keep pages of the current nb_ts object

Input:

- obj : An object of class nb_ts
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty a empty nb_ts object is returned.

Can also be a cellstr with the names to keep, or a logical array with length equal to the number of pages (datasets).

Output:

- obj : An nb_ts object with the pages specified in the input are kept.

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► keepVariables ↑

```
obj = keepVariables(obj,keptVar)
```

Description:

Keep variables from the current nb_ts object (letter size has something to say)

If some of the variables you specify in keptVar does not exist in the object, this will not have anything to say for the object. (The variables cannot be kept, because the variables do not exist.)

Input:

- obj : An object of class nb_ts
- keptVar : Must be a char or a cellstr

Keeps all the variable names, given in the char array or cellstr array keptVar, of current the nb_ts object.

NB! If keptVar is just one string and include a * as the last letter, this function will keep all the variable names, from the DB, that start with the letters which is in front of the * in the string.

i.e. obj = keepVariables(obj,'DPQ*');

Keep all the variables in the obj which has variable names starting with 'DPQ'.

If keptVar is just one string and include a * as the first letter, this function will keep all the variable names, from the DB, that include the letters which follows in the string.

i.e. obj = keepVariables(obj, '*shift');

Keeps all the variables in the obj which has variable names which include 'shift'.

Output:

- obj : An nb_ts object with the variables specified in input 'keptVar' kept, all the others are deleted.

Examples:

```
obj = keepVariables(obj, {'Var1','Var2'});
obj = keepVariables(obj, char('Var1','Var2'));
obj = keepVariables(obj, DB.variables);
```

Written by Kenneth S. Paulsen

► **ksdensity** ↑

```
obj = ksdensity(obj,varargin)
```

Description:

A method for creating an nb_ts object with the estimated kernel density of a sample. It can estimate a single probability density function, as well as recursive estimation.

Input:

- obj : An nb_ts object with the data.

Optional input:

- 'recursive' : True or false. If true, the CDF will be estimated recursively. Default is false.
- 'recStart' : Sets the start date for the recursive estimation. Default is the start date of the obj + 9 observations. As a string or as an object of class nb_date.
- 'recEnd' : Sets the end date for the recursive estimation. Default is the end date of the object. As a string or as an object of class nb_date.
- 'estStart' : Sets the start of the estimation sample. As a string or as an object of class nb_date.
- Rest of the optional input id given to the nb_ksdensity function.

Output:

- obj : A numIter x numVar nb_ts object where each observation point contains an nb_distribution object with the estimated parameters of the distribution.

Examples:

```
f    = nb_ts(randn(1000,1),'', '2010M1', {'v1'});
obj = f.ksdensity('recStart','2012M1','recEnd','2012M12', 'recursive',1);
```

See also:

[nb_ts.ecdf](#)

Written by Kenneth Sæterhagen Paulsen

► **kurtosis** ↑

`obj = kurtosis(obj,flag,outputType,dimension,varargin)`

Description:

Calculate the kurtosis of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their kurtosis.

The output can also be set to be a double or a nb_cs object consisting of the kurtosis values of the data.

Input:

- `obj` : An object of class nb_ts
- `flag` : > 0 : normalises by N-1 (Default)
> 1 : normalises by N
 - Where N is the sample length.
- `outputType` :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their kurtosis.
 - > 'double' : Get the kurtosis values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the kurtosis values as nb_cs object. The kurtosis will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- `dimension` : The dimension to calculate the kurtosis over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_tsObj = kurtosis(obj);
double    = kurtosis(obj,0,'double');
nb_csObj = kurtosis(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► **lag ↑**

```
obj = lag(obj,periods,varargin)
```

Description:

Lag the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_ts
- periods : Lag the data with this number of periods. Default is 1 period.

Optional input:

- 'cut' : true (cut sample to fit old sample length) or false (append new observations). Default is true.

Output:

- obj : An nb_ts object where the data laged by number of periods given by the input periods.

Examples:

```
obj = lag(obj,1);
```

Written by Kenneth S. Paulsen

► **lastObservation ↑**

```
obj = lastObservation(obj,number)
```

Description:

Get the last (real) observation(s) of the object.

Input:

- obj : An object of class nb_ts.
- number : Number of elements to return. Default is 1, i.e only the last.
As a integer.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► le ↑

a = le(a,b)

Description:

Test if the one object is less than or equal to (\leq) the other object elementwise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **lead** ↑

```
obj = lead(obj,periods,varargin)
```

Description:

Lead the data of the object. The input periods decides for how many periods.

Input:

- obj : An object of class nb_ts
- periods : Lead the data with this number of periods. Default is 1 period.

Optional input:

- 'cut' : true (cut sample to fit old sample length) or false (append new observations at the start). Default is true.

Output:

- obj : An nb_ts object where the data leaded by number of periods given by the input periods.

Examples:

```
obj = lead(obj,1);
```

Written by Kenneth S. Paulsen

► **log** ↑

```
obj = log(obj)
```

Description:

Take log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on logs.

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **log10** ↑

```
obj = log10(obj)
```

Description:

Take the common (base 10) log of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where the data are on (base 10) logs.

Examples:

```
obj = log10(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj. Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = log1p(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► log2 ↑

```
obj = log2(obj)
```

Description:

log2(obj) is the base 2 logarithm of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = log2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► lt ↑

```
a = lt(a,b)
```

Description:

Test if the one object is less than (<) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a < b;  
obj = 2 < b;  
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► map2Distribution ↑

```
obj = map2Distribution(obj,dist)  
obj = map2Distribution(obj,dist,varargin)
```

Description:

Map observation to the distributions given by dist.

The current distribution of the data is estimated using a kernel density estimator. The data must be strongly stationary. I.e. the unknown distribution from where the data has been drawn must not depend on time.

Input:

- obj : An object of class nb_dataSource.
- dist : An object of class nb_distribution with size 1 x size(obj,2).

Optional inputs:

- See the optional inputs of the nb_distribution.estimate function.

Output:

- obj : An object of class nb_dataSource.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **mavg** ↑

```
obj = mavg(obj,backward,forward,flag)
```

Description:

Taking moving avarage of all the timeseries of the nb_ts object

Input:

- obj : An object of class nb_ts
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average
- flag : If set to true the periods that does not have enough observations forward or backward should be set to nan. Default is false.

Output:

- obj : An nb_ts object storing the calculated moving average

Examples:

```
data = nb_ts(rand(50,2)*3,'','2011Q1',{'Var1','Var2'});  
mAvg10 = mavg(data,9,0); % (10 year moving average)
```

See also:

[nb_mavg](#)

Written by Kenneth S. Paulsen

► **max** ↑

```
obj = max(obj,outputType,dimension)
```

Description:

Calculate the max of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are beeing set to their max.

The output can also be set to be a double or a nb_cs object consisting of the max values of the data.

Input:

- obj : An object of class nb_ts
- outputType :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their max.
 - > 'double' : Get the max values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the max values as nb_cs object. The max will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the max over.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```
nb_tsObj = max(obj);
double   = max(obj,'double');
nb_csObj = max(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► mean ↑

```
obj = mean(obj,outputType,dimension,varargin)
```

Description:

Calculate the mean of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their mean.

The output can also be set to be a double or a nb_cs object consisting of the mean values of the data.

Input:

- obj : An object of class nb_ts

```

- outputType :

    > 'nb_ts'           : The result will be an object of class nb_ts
                           where all the non-nan values of all the
                           timeseries are being set to their mean.

    > 'nb_ts_scalar'   : The result will be an object of class nb_ts
                           where all the non-nan values of all the
                           timeseries are being set to their mean, but
                           where the dimension taking the mean over is
                           reduced to size 1. Not supported for dimension
                           1. The name of the variable/page will be
                           'mean'

    > 'double'          : Get the mean values as doubles, where
                           each column of the double matches the
                           variable location in the 'variables' property.
                           If the object consists of more pages the double
                           will also consist of more pages.

    > 'nb_cs'           : Get the mean values as nb_cs object.
                           The mean will when this option is used only be
                           calculated over the number of observations.
                           (I.e. dimension set to 1.)

- dimension : The dimension to calculate the mean over.

```

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more on the output from this method

Examples:

```

nb_tsObj = mean(obj);
double   = mean(obj,'double');
nb_csObj = mean(obj,'nb_cs');

```

Written by Kenneth S. Paulsen

► **meanDiff ↑**

```
obj = meanDiff(obj)
```

Description:

Construct level variables that has the mean change for each period.

Input:

- obj : An object of class nb_ts

Output:

- obj : A nb_ts object where the data has been updated.

Written by Kenneth S. Paulsen

► **meanGrowth** ↑

obj = meanGrowth(obj)

Description:

Construct level variables that has the mean growth for each period.

Input:

- obj : An object of class nb_ts

Output:

- obj : A nb_ts object where the data has been updated.

Written by Kenneth S. Paulsen

► **median** ↑

obj = median(obj,outputType,dimension,varargin)

Description:

Calculate the median of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are bbeing set to their median.

The output can also be set to be a double or a nb_cs object consisting of the median values of the data.

Input:

- obj : An object of class nb_ts

- outputType :

> 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are bbeing set to their median.

```

> 'double' : Get the median values as doubles, where
each column of the double matches the
variable location in the 'variables' property.
If the object consist of more pages the double
will also consist of more pages.

> 'nb_cs'   : Get the median values as nb_cs object.
The median will when this option is used only
be calculated over the number of observations.
(I.e. dimension set to 1.)

- dimension : The dimension to calcualate the median over.

```

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from
this method

Examples:

```

nb_tsObj = median(obj);
double   = median(obj,'double');
nb_csObj = median(obj,'nb_cs');

```

Written by Kenneth S. Paulsen

► **merge ↑**

```

obj = merge(obj,DB)
obj = merge(obj,DB,interpolateDate,method,rename,append,type)

```

Description:

Merge two nb_ts objects. Same as obj = [obj,DB]

Caution:

- It is possible to merge datasets with the same variables as long as they represent the same data or have different timespans.
- If the datsets has different number of datasets and one of the merged objects only consists of one, this object will add as many datasets as needed (copies of the first dataset) so they can be merged.
- If any of the datasets are of type real-time, they cannot contain the same variables.

- If one dataset is of type real-time, then the other dataset will need to only have one page. The other dataset will get the same recursive structure as the real-time dataset.
- If two real-time datasets are merge the last vintage of the two dataset cannot differ, when it comes to the endDate, by more than one period. If thy differ by one period it is assumed that this is the case for all vintages backward in time as well, i.e. the regged edge of the last period is perserved for all vintages.
- If two real-time datasets are merge and one of them have more vintages backward in time, the one with the fewest vintages will be expanded with the same number of vintages as the other dataset. The first avilable vintage then be used to construct these vintages.
- If two real-time datasets are merged you should first use the method cleanRealTime on both inputs!

Input:

- obj : An object of class nb_ts
- DB : An object of class nb_ts
- interpolateDate :
 - How to transform the database with the lowest frequency.
 - 'start' : The interpolated data are taken as given at the start of the periods. I.e. use 01.01.2012 and 01.01.2013 when interpolating data with yearly frequency. (Default)
 - 'end' : The interpolated data are taken as given at the end of the periods. I.e. use 31.12.2012 and 31.12.2013 when interpolating data with yearly frequency.
- method : The method to use when converting:
 - 'linear' : Linear interpolation
 - 'cubic' : Shape-preserving piecewise cubic interpolation
 - 'spline' : Piecewise cubic spline interpolation
 - 'none' : No interpolation. All data in between is given by nans (missing values) (Default)
 - 'fill' : No interpolation. All periods of the higher frequency get the same value as that of the lower frequency.
- rename : > 'on' : Renames the prefixes (default)
 > 'off' : Do not rename the prefixes

For more see the 'rename' option of the convert method.

- append : > 0 : If data is conflicting an error will occur.
default.
> 1 : If data is conflicting data of the first database will be used.
- type : The type of series to merge. Only when append is equal to 1.
 - > 'levelGrowth' : Merge level data with growth rates. Must be one period growth. The merged series will be in level.
 - > 'growthLevel' : Merge growth rates with level data. Must be one period growth. The merged series will be in growth rates.
 - > 'levelLevel' : Merge level series, where the merging is done in growth rates, but the end result is still in levels.
 - > '' : Standard merging.

If two real-time datasets are merge:

- > 'pages' : In this case the merging of the real-time databases is taken page by page starting from the last page. If some of the datasets have more pages than the other a error will be given.
- > '' : Default merging of real-time datasets. See the description part.

Output:

- obj : An nb_ts object where the datasets from the two nb_ts objects are merged

Examples:

```
obj = merge(obj,DB)
obj = obj.merge(DB)
```

See also:

[nb_ts.append](#), [nb_dataSource.merge2Series](#)

Written by Kenneth S. Paulsen

► **merge2Series ↑**

```
obj = merge2Series(obj,var1,var2,new,method,varargin)
```

Description:

Merge 2 series. The var2 series is appended to the var1 series from where it is ending.

Input:

- obj : An object of class nb_dataSource.
- var1 : A one line char with the name of the base series.
- var2 : A one line char with the name of the appended series.
- new : Name of the new variable. If given as empty there will not be created a new series, but instead the var1 variable will be set to the new merged series. Default is empty.
- method : Either 'level', 'diff', 'growth', 'leveldiff' or 'levelgrowth'. Default is 'level'. For the cases 'level', 'diff' or 'growth' it is assumed the both series are in levels. 'leveldiff' assumes the first series is in level and the second in nLags-difference. 'levelgrowth' assumes the first series is in level and the second in growth rates over nLags periods, i.e. $(x(t) - x(t-nLags))/x(t-nLags)$.

Optional input:

- 'nLags' : The number of lages used for the methods 'diff' and 'growth'. Default is 1.
- 'date' : The date from where to use the second variable. Either a date as string or a nb_date object. If empty the merge will take place where the first variable end + 1 period.

Output:

- obj : An object of class nb_dataSource.

Examples:

```
obj           = nb_ts.rand('2012Q1',10,2,3);
obj(end-1:end,1,:) = nan;

obj1 = merge2Series(obj,'Var1','Var2','Var3')
obj2 = merge2Series(obj,'Var1','Var2')
obj3 = merge2Series(obj,'Var1','Var2','','diff','nLags',1)
obj4 = merge2Series(obj,'Var1','Var2','','diff','nLags',4)
obj5 = merge2Series(obj,'Var1','Var2','','level','date','2014Q1')
```

See also:

[nb_ts.merge](#), [nb_data.merge](#), [nb_cs.merge](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **mergeContexts** ↑

```
obj = mergeContexts(varargin)
```

Description:

Merge real-time datasets that has the context dates stored in the dataNames property on the format 'yyyymmdd' and only represent one variable.

Caution: If the publishing calandar of one of the nb_ts objects is shorter than the other, the shortes of them will be expanded backwards with a quasi-real-time calandar. It will use the first publication date, and backcast the artifical publication dates by the time lag of this publication date. To construct the time series of these quasi-real-time data, it will use the first vintage, and recursivly remove one observation.

Caution : This method will break the link to the data source of updateable objects!

Optional input:

- varargin : Each object must be of class nb_ts on the specific format described in the Description of this method.

Output:

- obj : The updated set of contexts stored in a nb_ts object.

Written by Kenneth SÃ¶terhagen Paulsen

► **min** ↑

```
obj = min(obj,outputType,dimension)
```

Description:

Calculate the min of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are beeing set to their min.

The output can also be set to be a double or a nb_cs object consisting of the min values of the data.

Input:

```
- obj : An object of class nb_ts

- outputType :

    > 'nb_ts' : The result will be an object of class nb_ts
                  where all the non-nan values of all the
                  timeseries are being set to their min.

    > 'double' : Get the min values as doubles, where
                  each column of the double matches the
                  variable location in the 'variables' property.
                  If the object consists of more pages the double
                  will also consist of more pages.

    > 'nb_cs' : Get the min values as nb_cs object.
                  The min will when this option is used only
                  be calculated over the number of observations.
                  (I.e. dimension set to 1.)

- dimension : The dimension to calculate the min over.
```

Output:

```
- obj : See the input outputType for more one the output from
      this method
```

Examples:

```
nb_tsObj = min(obj);
double   = min(obj,'double');
nb_csObj = min(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► minus ↑

```
obj = minus(obj,DBOrNum)
```

Description:

Binary subtraction (-).

Caution: Will subtract the data of the corresponding variables from the two objects. All the variables not in common will result in series with nan values.

When one of the inputs are a scalar this function will subtract each element of the data by this number or each element of the data will be subtracted from the scalar.

Input:

- a : An object of class nb_ts or a scalar
- b : An object of class nb_ts or a scalar

Output:

- obj : An nb_ts object where all the variables are from the first object are substracted from the other.

Or

An nb_ts object where the scalar are substracted from all the elements of the input object

Or

An nb_ts object where all the elements of the input object are substracted from scalar.

Examples:

```
obj = obj - anotherObj;
obj = obj - 2;
obj = 2 - obj;
```

See also:

[nb_ts.plus](#), [nb_ts.callop](#), [nb_ts.callfun](#)

Written by Kenneth S. Paulsen

► **mldivide** ↑

```
obj = mldivide(a,b)
```

Description:

Matrix left division (\). See examples.

Input:

- a : An object of class nb_ts
- b : A scalar, a string or an nb_date object.

Output:

- obj : An nb_ts object.

Examples:

```
obj = nb_ts.rand('2012Q1',10,2)
obj = obj\1 % 1/obj
obj = obj\'2012Q1' % '2012Q1'/obj
obj = obj\'Var1' % 'Var1'/obj
obj = obj\obj.startDate % obj.startDate/obj
```

See also:

[nb_ts.mrdivide](#), [nb_ts.rdivide](#)

Written by Kenneth S. Paulsen

► **mode** ↑

```
obj = mode(obj,outputType,dimension,varargin)
```

Description:

Calculate the mode of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their mode.

The output can also be set to be a double or a nb_cs object consisting of the mode values of the data.

Input:

- obj : An object of class nb_ts
- outputType :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their mode.
 - > 'double' : Get the mode values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the mode values as nb_cs object. The mode will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the mode over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_tsObj = mode(obj);
double    = mode(obj,'double');
nb_csObj = mode(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► **mpower** ↑

```
obj = mpower(a,b)
```

Description:

Matrix power (^). Defined when raised by a scalar, a string which either represent a variable or date or a nb_date object.

Input:

- a : An object of class nb_ts.
- b : A scalar, a string or an nb_date object.

Output:

- obj : An nb_ts object.

Examples:

```
obj  = nb_ts.rand('2012Q1',10,2)
obj1 = obj^'2012Q1'
obj2 = obj^'Var1'
obj3 = obj^obj.startDate
```

See also:

[nb_ts.power](#)

Written by Kenneth S. Paulsen

► **mrddivide** ↑

```
obj = mrddivide(a,b)
```

Description:

Matrix right division (/). See examples.

Input:

- a : An object of class nb_data, a scalar number, a string representing a variable name or a date, or a nb_date object.
- b : An object of class nb_data, a scalar number, a string representing a variable name or a date, or a nb_date object.

Output:

- obj : An nb_ts object.

Examples:

```
obj = nb_ts.rand('2012Q1',10,2)
obj1 = obj/1
obj2 = obj/'2012Q1'
obj3 = obj/'Var1';
obj4 = obj/obj.startDate
obj5 = 1/obj
obj6 = '2012Q1'/obj
obj7 = 'Var1'/obj;
obj8 = obj.startDate/obj
```

See also:

[nb_ts.mldivide](#), [nb_ts.rdivide](#)

Written by Kenneth S. Paulsen

► mstd ↑

```
obj = mstd(obj,backward,forward)
```

Description:

Taking moving standard deviation of all the timeseries of the nb_ts class

Input:

- obj : An object of class nb_ts
- backward : Number of periods backward in time to calculate the moving std
- forward : Number of periods forward in time to calculate the moving std

Output:

- obj : An nb_ts object storing the calculated moving standard deviation

Examples:

```
data = nb_ts(rand(50,2)*3,'','2011Q1',{'Var1','Var2'});
```

```
mstd10 = mstd(data,9,0); % (10 year moving std)
```

Written by Kenneth S. Paulsen

► mtimes ↑

```
obj = mtimes(obj,Num)
```

Description:

Matrix multiplication (*) Defined for multiplication with a constant, a string which either represent a variable or date or an nb_date object.

Input:

- obj : An object of class nb_ts or scalar
- b : An object of class nb_ts or a scalar, a string or an nb_date object.

Output:

- obj : An nb_ts object where the number (date or variable as a vector) is multiplied with all the elements of the input objects data

Examples:

```
obj = 2*obj;
obj = obj*2;
obj = obj*'2012Q1'
obj = obj*'Var1'
obj = obj*obj.startDate
```

See also:

[nb_ts.times](#)

Written by Kenneth S. Paulsen

► **nan** ↑

```
obj = nb_ts.nan()
obj = nb_ts.nan(start,obs,vars,pages,sorted)
```

Description:

Create an nb_ts object with all data set to nan.

Input:

- start : The start date of the created nb_ts object.
- obs : The number of observation of the created nb_ts object.
- vars : Either a cellstr with the variables of the nb_ts object, or a scalar with the number of variables of the nb_ts object.
- pages : Number of pages of the nb_ts object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_ts object.

Examples:

```
obj = nb_ts.nan('1',10,{'Var1','Var2'});
obj = nb_ts.nan('2012Q1',2,2);
obj = nb_ts.nan('2012Q1',2,2,10);
```

See also:

[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► **nan2var** ↑

```
obj = nan2var(obj,testedVariables,assignVariable)
```

Input:

- obj : An object of class nb_ts
- testedVariables : A cellstr with the variables you are setting equal to the assignVariable for the nan values found.
- assignVariable : A string with the name of the variable you are assigning the found nan values.

Output:

- obj : An nb_ts object where all the nan values of the variables in the testedVariables input is set to the matching value of the variable given by the assignVariable input.

Examples:

```
obj = nb_ts([1,2;nan,1;nan,3],'', '2012', {'Var1','Var2'});  
nan2var(obj,{'Var1'},'Var2')
```

```
ans =
```

'Time'	'Var1'	'Var2'
'2012'	[1]	[2]
'2013'	[1]	[1]
'2014'	[3]	[3]

Written by Kenneth S. Paulsen

► ne ↑

```
a = ne(a,b)
```

Description:

Test if the one object is not equal to ($\sim=$) the other object elemetswise and return a nb_dataSource object where each element are 1 if true, otherwise 0.

Caution: The objects must have the same dimension and variables!

It is also possible to test each element of an object to a scalar number.

Input:

- a : An object of class nb_dataSource or a scalar number.
- b : An object of class nb_dataSource, but must be of same subclass as a, or a scalar number.

Output:

- a : An nb_dataSource object where the data element are logical.

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **not** ↑

```
a = not(a)
```

Description:

The not operator (~).

Input:

- a : An object of class nb_dataSource.

Output:

- a : An object of class nb_dataSource. Where the not operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = ~a;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **objSize** ↑

```
varargout = objSize(obj,dim)
```

Description:

Return the size(s) of the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- dim : - 1 : Number of observations
- 2 : Number of variables
- 3 : Number of datasets (pages)

Output:

- varargout : See the examples below

Examples:

```
dim = objSize(obj)

Where dim is 1 x 2 double where the first element
is the number of observations/types and the second element is
the number of variables
```

```
[dim1, dim2] = objSize(obj)

[dim1, dim2, dim3] = objSize(obj)

dim1 = objSize(obj,1)
dim2 = objSize(obj,2)
dim3 = objSize(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **ones** ↑

```
obj = nb_ts.ones()
obj = nb_ts.ones(start,obs,vars,pages,sorted)
```

Description:

Create an nb_ts object with all data set to 1.

Input:

- start : The start date of the created nb_ts object.
- obs : The number of observation of the created nb_ts object.
- vars : Either a cellstr with the variables of the nb_ts object, or a scalar with the number of variables of the nb_ts object.
- pages : Number of pages of the nb_ts object.
- sorted : true or false. Default is true.

Output:

- obj : An nb_ts object.

Examples:

```
obj = nb_ts.ones('1',10,['Var1','Var2']);
obj = nb_ts.ones('2012Q1',2,2);
obj = nb_ts.ones('2012Q1',2,2,10);
```

See also:

[nb_ts](#)

► **or** ↑

```
a = or(a,b)
```

Description:

The and operator (|).

Caution: The objects must have the same dimension and variables!

Input:

- a : An object of class nb_dataSource.
- b : An object of class nb_dataSource, but must be of same subclass as a!

Output:

- a : An object of class nb_dataSource. Where the or operator has evaluated all the data elements of the object.
The data will be a logical matrix.

Examples:

```
a = a | b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **packing** ↑

```
obj = packing(obj,packages,varargin)
```

Description:

Package variables by applying the wanted function. Default is summin (sum).

Input:

- obj : An object of class nb_ts.
- packages : A 2 x N cell matrix with groups of variables and the names of the groups. If you have not listed a variable it will be grouped in a group called 'Rest' (as long as

'includeRest' is not set to false).

Must be given in the following format:

```
{'group_name_1','...','group_name_N';
 name_1           ,...,name_N}
```

Where name_x must be a string with a name of the variable or a cellstr array with the variable names. E.g 'Var1' or {'Var1','Var2'}.

Optional inputs:

- 'includeRest' : Give false to not include the 'Rest' variable, which will represent all the variable that has not been packed.
- 'func' : Either a function_handle or a one line char with the name of the function to use. The first input to this function is a nObs x nVar double, the second is the dimension (which is always 2). Examples; 'sum', 'prod', @sum or @prod.

Output:

- obj : A object of class nb_ts.

Examples:

```
data      = nb_ts.rand('2012Q1',10,4);
packages = {'group1','group2';
            {'Var1','Var2'},{'Var3'}};

dataP1 = packing(data,packages)
dataP2 = packing(data,packages,'includeRest',false)
dataP3 = packing(data,packages,'func',@prod)
```

See also:

[nb_ts.createVariable](#), [nb_data.createVariable](#), [nb_cs.createVariable](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **parcorr ↑**

```
obj = parcorr(obj,numLags)
[obj,lBound,uBound] = parcorr(obj,numLags,errorBound,alpha)
```

Description:

Compute the sample partial autocorrelation function (PACF) of all the variables stored in the nb_ts object, and return a nb_data object with the results.

Reference:

[1] Box, G. E. P., G. M. Jenkins, and G. C. Reinsel. Time Series Analysis: Forecasting and Control. 3rd edition. Upper Saddle River, NJ: Prentice-Hall, 1994.

Input:

- obj : An object of class nb_ts.

Caution : This method does not handle nan or infinite for any of the stored data of the nb_ts object!

- numLags : Positive integer indicating the number of lags of the PACF to compute. If empty or missing, the default is to compute the PACF at lags 1. Since PACF is symmetric about lag zero, negative lags are ignored. Must be a scalar.

- errorBound : A string. The data must be stationary. Either:

> '' : No errorbounds calculated.

> 'asymtotic' : Using asymptotically derived formulas.

> 'paramBootstrap' : Calculated by estimating the data generating process. And use the fitted model to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.

> 'blockBootstrap' : Calculated by blocking the observed series and using these blocks to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.

> 'copulaBootstrap' : The observed series is boostraped based on a copula approach. See Paulsen (2017).

- alpha : Significance value. As a scalar. Default is 0.05.

Optional inputs:

- 'maxAR' : As an integer. See the nb_arima function. Defualt is 3.

- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- obj : An nb_data object with the partial autocorrelations stored in the dataset 'Autocorrelations'.
If the errorbounds are to be calculated they are stored in the datasets 'Error bound (lower)' and 'Error bound (upper)'

Examples:

```
obj = parcorr(obj)
obj = parcorr(obj,10)
```

See also:

[nb_ts](#), [nb_data](#), [corr](#), [nb_parcorr](#)

Written by Kenneth Sæterhagen Paulsen

► [pca ↑](#)

```
F = pca(obj)
[F,LAMBDA,R,varF,expl,c,sigma,e,Z] = pca(obj,r,method,varargin)
```

Description:

Principal Component Analysis of the data of the object.

Input:

- obj : An object of class nb_ts.
 - r : The number of principal component. If empty this number will be found by the Bai and Ng (2002) test, see the optional option crit below, i.e choose the CT part of the selection criterion; $\log(V(ii,F)) + CT$. Where;
- $$V(ii,F) = \sum(\text{diag}(e_{ii}' * e_{ii}/T))/ii$$
- and e_{ii} is the residual when ii factors are used

```

- method : Either 'svd' or 'eig'. Default is 'svd'.

    > 'svd' : Uses single value deomposition to construct the
               principal components. This is the numerically best
               way, as the 'eig' method could be very unprecise!

    > 'eig' : Uses the eigenvalue decomposition approach instead

```

Optional inputs:

```

- 'crit' : You can choose between the follwing selection criterion

    > 1: log(NT/NT1)*ii*NT1/NT;

    > 2: (NT1/NT)*log(min([N;T]))*ii;

    > 3: ii*log(GCT)/GCT;

    > 4: 2*ii/T;

    > 5: log(T)*ii/T;

    > 6: 2*ii*NT1/NT;

    > 7: log(NT)*ii*NT1/NT; (default)

where NT1 = N + T, GCT = min(N,T), NT = N*T and ii = 1:rMax

- 'rMax' : The maximal number of factors to test for. Must be less than
            or equal to N. Default is N.

- 'trans' : Give 'demean' if you want to demean the data in the matrix
            X. Default. Give 'standardize' if you want to standardise the
            data in the matrix X. Gice 'none' to drop any kind of
            transformation of the data in the matrix X.

Caution: To get back X when 'demean' is given you need to add
          the constant terms in c. I.e;

X = c(ones(T,1),1) + F*LAMBDA + e, e ~ N(0,R) (1')

To get back X when 'standardise' you need to add
          the constant term and multiply with sigma. I.e.

X = c(ones(T,1),:) + F*LAMBDA.*sigma(ones(T,1),:) +
    eps, eps ~ N(0,R*) (1*)

Be aware that the eps differ from e, as e is the
          residual from the standardised regression.

'output' : A string with the wanted output. Either 'F', 'LAMBDA',
            'R', 'varF', 'expl', 'c', 'sigma', 'e'. See below for the
            matching output.

Caution: If provided it will be the only output from this
          function. I.e. return as F.

- 'unbalanced' : true or false. Set to true to allow for unbalanced
                 dataset. Default is false. If false all rows of the

```

dataset containing nan value are removed from calculations.

Output:

- F : The principal component, as a nb_ts object.
- LAMBDA : The estimated loadings, as a nb_cs object
- R : The covariance matrix of the residual in (1), as a nb_cs object.
- varF : As a nb_cs object. Each column will be the variance of the corresponding column of F.
- expl : The percentage of the total variance explained by each principal component. As a 1 x r nb_cs object
- c : Constant in the factor equation. nb_cs object with size 1 x nvar.
- sigma : See 'trans'. nb_cs object with size 1 x nvar.
- e : Residual of the equation (1) above. As a nb_ts object.
- Z : Normalized (and rebalanced) version of X.

Example:

```
load hald;
ingredients = nb_ts(ingredients,'',1,{'
    tricalcium_aluminate',...
    'tricalcium_silicate',...
    'tetracalcium_aluminoferrite',...
    'beta_dicalcium_silicate'});
[F,LAMBDA,R,varF,expl] = pca(ingredients)
```

See also:

[nb_pca](#)

Written by Kenneth SÃ¶terhagen Paulsen

► [pcn](#) ↑

```
obj = pcn(obj);
obj = pcn(obj,lag,stripNaN)
```

Description:

Calculate log approximated percentage growth. Using the formula
 $(\log(t+lag) - \log(t)) * 100$

Input:

- obj : An object of class nb_ts
- lag : The number of lags. Default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An nb_ts object with the log approximated percentage growth data stored.

Examples:

```
obj = pcn(obj); % period-on-period log approx. growth
obj = pcn(obj,4); % 4-periods log approx. growth
```

See also:

[growth](#), [egrowth](#), [ipcn](#)

Written by Kenneth S. Paulsen

► **percentiles** ↑

```
obj = percentiles(obj,perc)
```

Description:

Take the percentile over the third dimension of the data of the object.

Input:

- obj : An object of class nb_ts
- perc : A double with the wanted percentiles of the data. E.g. 0.5 (median), or [0.05,0.15,0.25,0.35,0.5,0.65,0.75,0.85,0.95].

Output:

- obj : A nb_ts object with the percentiles stored as different pages of the object. See the dataNames property for the ordering.

Written by Kenneth Sæterhagen Paulsen

► **permute** ↑

```
obj = permute(obj,variables)
```

Description:

Switch the second and third dimension of the nb_ts, nb_cs or nb_data object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- variables : A cellstr with size 1 x numberOfDatasets. Default is to use dataNames property. I.e. when not provided or if empty.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► plot ↑

```
plot(obj,varargin)
plotter = plot(obj,varargin)
```

Description:

Plot the data of the nb_dataSource object.

Caution: If the data of the object is of class nb_distribution this method will redirect to plotDist.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- plotType : A string with either:
 - > 'graph' (Default)
 - > 'graphSubPlots'
 - > 'graphInfoStruct'The name of the graphing method of the class nb_graph_ts, nb_graph_cs or nb_graph_data.
- Caution: This is an optional input. If not given it is assumed that the optional input pairs are starting from the second input to this function.
- varargin : Same as the input to the method set() of the nb_graph_ts, nb_graph_cs or nb_graph_data class.

I.e. ..., 'inputName', inputValue, ...

Caution: If the data of the object is of class nb_distribution, see nb_ts.plotDist for the optional input pairs supported.

Output:

Plotted output

Examples:

```
plot(obj)  
plot(obj,'graphSubPlots')  
plot(obj,'','startGraph','2008Q1','endGraph','2011Q2')  
same as  
plot(obj,'graph','startGraph','2008Q1','endGraph','2011Q2')
```

See also:

[nb_graph_ts](#), [nb_ts.plotDist](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **plotDist** ↑

f = plotDist(obj, varargin)

Description:

A method for plotting the data of an nb_ts object when it is of type nb_distribution. It allows for both 2D and 3D plots.

Input:

- obj : An nb_ts object.

Optional input:

- '3d' : Either true or false. If true, the method will create a 3D-plot. Otherwise it will return 2D plots. Default is false.
- 'type' : What type of distribution to plot, either cdf or pdf. As string.
- 'function' : Either 'surf' or 'mesh', depending on how you want the

surface of the 3D-plot to look like.

- 'angle' : Allows for adjusting the angle of the X-labels. Default is 60.
- 'startGraph' : Sets the start date of the plot. Default is the start date of the data. As a string or as an object of class nb_date.
- 'endGraph' : Sets the end date of the plot. Default is the end date of the data. As a string or as an object of class nb_date.
- Other optional inputs: See the nb_graph_data class.

Output:

- f : Either a MATLAB figure or a nb_graph_data object.

Examples:

First create some data:

```
obj = nb_ts(randn(1000,2),'', '2010M1', {'v1','v2'})
```

Recursively estimate a pdf of the data:

```
rD = obj.ksdensity('recStart','2012M1','recEnd','2030M1', 'recursive',1)
```

Create a 3D-plot

```
rD.plotDist('3d',1)
```

See also:

[nb_ts.plot](#)

Written by Kenneth Sæterhagen Paulsen

► **plus ↑**

```
obj = plus(obj,DBOrNum)
```

Description:

Binary addition (+).

Caution: Will add the data of the corresponding variables from the two objects. All the variables not in common will result in series with nan values.

When one of the inputs are a scalar this function will add each element of the data by this number.

Input:

- a : An object of class nb_ts or a scalar
- b : An object of class nb_ts or a scalar

Output:

- obj : An nb_ts object where all the variables from the two objects are added.

Or

An nb_ts object where the scalar are added to all elements of the given nb_ts object

Examples:

```
obj = obj + anotherObj;  
obj = obj + 2;  
obj = 2 + obj;
```

See also:

[nb_ts.minus](#), [nb_ts.callop](#), [nb_ts.callfun](#)

Written by Kenneth S. Paulsen

► **pow2** ↑

```
obj = pow2(obj)
```

Description:

pow2(obj) raises the number 2 to the power of the elements of obj

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = pow2(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **power** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^).

If both inputs are nb_ts objects this method will raise the data of the first object with the data from the second object. This will only be done for the variables which are found to be in both nb_ts objects. The rest of the variables will get nan values.

Input:

- obj : An object of class nb_ts or a scalar
- DBOrNum : An object of class nb_ts or a scalar

Output:

- obj : An nb_ts object where the data from th one object are raised with the data from the other object

or

An nb_ts object where all the elements are raised by the scalar DBOrNum.

or

An nb_ts object where the scalar obj are raised by all the elements of the object DBOrNum.

Examples:

```
obj = a.^b;
obj = a.^2;
obj = 2.^a;
```

See also:

[nb_ts.mpower](#), [nb_ts.callop](#), [nb_ts.callfun](#)

Written by Kenneth S. Paulsen

► **q2y** ↑

```
obj = q2y(obj)
```

Description:

Will for each quarter calculate the cumulative sum over the last 4 quarter (Including the present). The frequency of the return object will be quarterly.

Input:

- obj : An object of class nb_ts

Output:

- obj : An nb_ts object where the data are the sum over the last 4 quarters.

Written by Kenneth S. Paulsen

► **rand** ↑

```
obj = nb_ts.rand(start,obs,vars,pages,dist,sorted)
```

Description:

Create an nb_ts object with all data set to random number.

Input:

- start : The start date of the created nb_ts object.
- obs : The number of observation of the created nb_ts object.
- vars : Either a cellstr with the variables of the nb_ts object, or a scalar with the number of variables of the nb_ts object.
- pages : Number of pages of the nb_ts object.
- dist : A nb_distribution object to draw from.
- sorted : true or false. Default is true.

Output:

- obj : An nb_ts object.

Examples:

```
obj = nb_ts.rand('1',10,['Var1','Var2']);
obj = nb_ts.rand('2012Q1',2,2);
obj = nb_ts.rand('2012Q1',2,2,10,nb_distribution('type','normal'));
```

See also:

[nb_ts](#)

Written by Kenneth Sætherhagen Paulsen

► **rdivide** ↑

`obj = rdivide(obj,DB)`

Description:

Right element-wise division (./)

Do right element-wise division of the matching variables of the two nb_ts objects. If some variables are not found to be in both objects, the resulting object will have timeseries with nan values only for these variables.

Input:

- a : An object of class nb_ts
- b : An object of class nb_ts

Output:

- obj : An object of class nb_ts, where the data are the result of the element-wise division.

Examples:

`obj = obj./DB;`

See also:

[nb_ts.mldivide](#), [nb_ts.mrdivide](#), [nb_ts.callop](#), [nb_ts.callfun](#)

Written by Kenneth S. Paulsen

► **reIndex** ↑

`obj = reIndex(obj,date,baseValue)`

Description:

Reindex the data of the object to a new date. It is also possible to reindex the timeseries to a date with lower frequency. E.g. reindex quarterly data to be on average 100 in a given year.

Input:

- obj : An object of class nb_ts
- date : The date at which the data should be reindexed to.
(Reindexed to baseValue, default is 100)
- baseValue : A number to re-index the series to. Default is 100.

Output:

- obj : An object of class nb_ts where all the objects timeseries are reindexed to 100 at the given date

Examples:

```
obj = reIndex(obj,'2012Q2')
```

Written by Kenneth S. Paulsen

► real ↑

```
obj = real(obj)
```

Description:

real(obj) returns the real part of the elements in the nb_cs object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = real(obj);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► realTime2RecCondData ↑

```
out = realTime2RecCondData(obj,nSteps,freq,nowcast,vintages)
```

Description:

Takes real time data and converts it into recursive conditional data, with the number of steps of forecast that you want.

Input:

- obj : A nb_ts object.
- nSteps : The number of steps that you want to include in your recursive dataset.
- freq : The frequency of your data. Must be either 1, 2, 4, 12 or 365.
- nowcast : Either 0 or 1. If true you will include todays estimate as your first forecast. Default is true.
- vintages : Give false if the dataNames property do not store the vintage dates as is the case for data returned by nb_fetchRealTimeFromFame. Be aware that this assumes that each new page only give one and only one new observation.

Give 2 if the dataNames property store the end date of history for each vintage (instead of a vintage). This is the case if the nb_fetchRealTime is used.

Output:

- out : A nb_data object

See also:

[nb_fetchRealTime](#), [nb_fetchRealTimeFromFame](#)

Written by Tobias Ingebrigtsen

► reallog ↑

obj = reallog(obj)

Description:

Take the real natural logarithm of the data stored in the object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = reallog(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **realpow** ↑

obj = realpow(obj,DBOrNum)

Description:

Same as nb_ts.power.

See also:

[nb_ts.power](#)

Written by Kenneth S. Paulsen

► **realsqrt** ↑

obj = realsqrt(obj)

Description:

realsqrt(obj) is the square root of the elements of obj.
An error is produced if obj contains negative elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = realsqrt(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► rebalance ↑

```
obj = rebalance(obj,methods)
```

Description:

Rebalance the dataset with a spesific method for each variable.

Caution: If a variable is not assign a spesific method. A AR(1) model with automatic selection of the level of intergration is used to backcast and forecast trailing and leading nan, while a spline method is used to fill in the missing observations in-sample. See nb_arRebalance and spline for more.

Input:

- obj : An T x nVars object of class nb_ts. Can only have one page.

- methods : A nVars x 3 cell array. First column must be the variable name, the second column must be the function to backcast and forecast trailing and leading nan, while the third column must be the function to fill in for nan values in-sample.

E.g. {'Var1',@(x)nb_arRebalance(x),@(x)nb_spline(x)}

If 'all' is set as the variable name, the selected options will be applied to all variables.

If not given or given as empty the default methods will be used.

Output:

- obj : A T x nVars object of class nb_ts.

See also:

[nb_arRebalance](#), [nb_spline](#)

Written by Kenneth Sæterhagen Paulsen

► recursiveExtrapolate ↑

```
out = recursiveExtrapolate(obj,variables,nSteps,startDate)
out = recursiveExtrapolate(obj,variables,nSteps,startDate,varargin)
```

Description:

Recursively extrapolate last of observation of variables with the wanted number of periods.

Caution: This method only works on a single paged nb_ts object or objects that return true using the isRealTime method.

Input:

- obj : An object of class nb_ts
- variable : A string with the name of the variable to extrapolate, or a cellstr of the variables to extrapolate. Default is to extrapolate all variables.

Caution: When 'method' is set to copula only the selected variables are included in the calculation of the extrapolated values.

- nSteps : Must be an integer with the number of periods to extrapolate.
- startDate : Start date of recursive extrapolation. Either as a char or an object of which is of a subclass of the nb_date class. This option is only used when the dataset is not real-time.

Caution : If all other than the 'end' option is used for the 'method' option. This options must allow for a proper number of degrees of freedom for estimating the model to use to extrapolate.

Optional inputs:

- 'method' :
 - > 'end' : Extrapolate series by using the last observation of each series. I.e. a random walk forecast.
 - > 'copula' : This approach tries to estimate the unconditional marginal distribution of each series in the dataset and the (auto)correlation structure that describe the connection between these series. Some series may have more observations than others!

Caution : This method works when the unconditional distribution from which the series are assumed to be drawn from are the same over time. E.g. the series must have the same volatility over the period. The second assumption is that all the series are stationary. Use the 'check' input to test for non-stationarity by a ADF test for unit-root. If it is found to have a unit root the series will be differenced when forecasted, and then transformed back to level after that step.

- > 'ar' : Extrapolate each series individually using a fitted

AR model.

```
> 'var'      : Extrapolate all series using a fitted VAR model. Max  
               number of variables is set to 10 for this option.  
  
> 'actual'   : Extrapolate with actual data. This will lead to  
               nans for the last periods!  
  
- 'alpha'     : Significance level of ADF test for stationary series.  
               Only an option for the 'copula' method.  
  
- 'check'     : Check for stationary series. If not found to be  
               stationary the series will be differenced and  
               extrapolated with this transformation before transformed  
               back to its old level. Only an option for the 'copula'  
               method.  
  
- 'nLags'     : Number of lags to include when calculating the  
               correlation matrix used by the copula. I.e. the number  
               of historical observations to condition on when forming  
               the conditional correlation matrix.  
  
- 'draws'     : The number of draws used to estimate the mean, when  
               using the 'copula' method.  
  
- 'constant'  : Give true to include constant in the AR models used  
               to extrapolate the individual series. Default is false.
```

Output:

```
- out         : A nb_ts object with the recursively extrapolated series.
```

Examples:

```
obj = nb_ts.rand('1900',50,2,1);  
obj = recursiveExtrapolate(obj,'Var1',4,'1900');
```

Written by Kenneth S. Paulsen

► **release ↑**

```
obj = release(obj,num)
```

Description:

Get a specific realease from a real-time dataset.

```
Caution : cleanRealTime should be ran to secure that this method does  
          not throw an error.
```

Input:

- obj : An object of class nb_ts representing real-time data.
- num : An integer larger than 0 with the release number.

Output:

- obj : An object of class nb_ts.

See also:

[nb_ts.cleanRealTime](#), [nb_readExcelMorePages](#), [nb_fetchRealTimeFromFame](#)
[nb_ts.getRelease](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations ↑**

obj = removeObservations(obj,numPer,vars)

Description:

Remove the number of wanted observation from each series, or a subset of variables. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_ts object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.
- vars : A cellstr with the variables to perform the method on. Can be empty, in which case the method will be performed on all variables in the object, which is the default.

Output:

- obj : A nb_ts object

Examples:

```
obj = nb_ts.rand(1,4,3);
obj = removeObservations(obj,1)
obj = removeObservations(obj,1,['Var1','Var2'])
```

Written by Kenneth Sæterhagen Paulsen

► **rename ↑**

```
obj = rename(obj,type,oldName,newName)
```

Description:

Makes it possible to rename variables and datasets. For objects of class nb_cs types is also possible to rename.

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variable', but not when the input is 'dataset' or 'types'.

Input:

- obj : An object of class nb_dataSource
- type : Either:
 - 'variable' : To rename a variable(s)
 - 'dataset' : To rename a dataset(s) (The only input that is supported for nb_cell)
 - 'types' : To rename a type(s) (only nb_cs)

- oldName : Case 1;

> The old variable/dataset/type name, as a string

Case 2:

> The string which should be replaced in all the variable names of the database. And reorder things afterwards. This input must end with a *.

- newName : Case 1;

> The new variable/dataset/type name, as a string

Case 2;

> The string which should replace the old string part given by the input 'oldName'.

Output:

- obj : An nb_dataSource object with the renamed variable(s), type(s) or dataset(s).

Examples:

```
obj = nb_cs([2,2],'test',{'First'},{'Var1','Var2'});  
obj = obj.rename('variable','Var1','Var3');  
obj = obj.rename('variable','Var*','N_Var');  
obj = rename(obj,'variable',{'Var1','Var2'},{'VAR1','VAR2'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **renameMore** ↑

```
obj = renameMore(obj,type,oldNames,newNames)
```

Description:

Makes it possible to rename more variables, datasets and types (only nb_cs).

Caution : It will resort the variables and the data accordingly when the input type is equal to 'variables', but not when the input is 'datasets'.

Caution : This is the same as looping over nb_dataSource.rename, but more efficient.

Caution : * syntax is not supported as is the case for nb_dataSource.rename

Input:

- obj : An object of class nb_dataSource.
- type : Either:
 - 'variables' : To rename a variables
 - 'datasets' : To rename a datasets (The only input that is supported for nb_cell)
 - 'types' : To rename a type (Only nb_cs)
- oldNames : The old variable/dataset/type names, as a cellstr.
- newName : The new variable/dataset/type names, as a cellstr.

Output:

- obj : An nb_dataSource object where the variable(s)/dataname(s)/type(s) of the object given as input is/are renamed.

Examples:

```
obj = nb_cs([22,24;27,28],'Dataset1',{'Exp','Imp'},{'Nor','Swe'})  
obj = renameMore(obj,'dataset',{'Dataset1'},{'tradeBal'});  
obj = renameMore(obj,'variable',{'Nor','Swe'},{'Norway','Sweden'});  
obj = renameMore(obj,'type',{'Exp','Imp'},{'Export','Import'});
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **reorder ↑**

```
obj = reorder(obj,newOrder,type)
```

Description:

Re-order the variables/types/datasets of the object.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- newOrder : A cellstr with the new order of the variables/types/datasets.
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **ret ↑**

```
obj = ret(obj)
obj = ret(obj,nlag)
obj = ret(obj,nlag,skipNaN)
```

Description:

Calculate return, using the formula: $x(t)/x(t-lag)$ of all the timeseries of the nb_ts object.

Input:

- obj : An object of class nb_ts
- nlag : The number of lags in the return formula, default is 1.
- skipNaN : -1 : Skip nan while using the ret operator. (E.g. when dealing with working days.)
 - 0 : Do not skip nan values. Default.

Output:

- obj : An nb_ts object with the calculated timeseries stored.

Examples:

```
obj = ret(obj);
obj = ret(obj,4);
```

See also:

[nb_ts](#)

► **rlag** ↑

```
obj = rlag(obj,t,endDate)
```

Description:

Append data with use of the rolling lag operator

Input:

- obj : An object of class nb_ts
- t : The number of lags to use for the rolling window
- endDate : The new end date

Output:

- obj : An object of class ts

Examples:

```
obj = nb_ts(rand(20,1),'',1,'Var1')
obj = rlag(obj,4,'30')
```

► **round** ↑

```
obj = round(obj,value,startDate,endDate,variables)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_ts object
- value : The rounding value. As a double. Default is 1.
- startDate : The start date of the rounding. As a date string or a nb_date object. Can be empty. Default is the start date of the data.

- endDate : The end date of the rounding. As a date string or a nb_date object. Can be empty. Default is the end date of the data.
- variables : The variables to round. Can be empty. Default is to round all variables.

Output:

- obj : A nb_ts object with the rounded numbers.

Written by Kenneth SÃ¶terhagen Paulsen

► **saveDataBase ↑**

saveDataBase(obj, varargin)

Description:

Save data of the object to (a) file(s)

If no optional input is given, i.e. obj.saveDataBase(). The data of the object is save down to xlsx file(s). Each dataset of the object is saved down to a excel spreadsheet with the name of the dataset (Taken from the 'dataNames' property)

Input:

- obj : An object of class nb_ts

Optional input (..., 'propertyName', PropertyValue, ...):

- 'saveName' : A string with the wanted name of the saved file.
- 'ext' : > 'xlsx' : Saves the object down to a excel spreadsheet. Default
 - > 'matold' : Saves the data down to a mat file. The data is transformed into a structure, with the variables as the fieldnames, and the data as its fields. If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method with input old set to 1.
Must be combined with the optional input 'saveName'.
 - > 'mat' : Saves the data down to a mat file. The data is transformed into a structure, with the variables stored in the field 'variables', while the data of the variables will be saved as its fields with generic

names ('Var1','Var2' etc). If the data consist of more pages, each field will consist of the same number of pages. See the toStructure method.

Must be combined with the optional input 'saveName'.

> 'mats' : The object is saved down as a struct using the nb_ts.struct function.

> 'txt' : Saves the data down to a txt file.

- 'append' : > 1 : Saves all the datasets (pages of the data) to one excel spreadsheet, but in separate worksheets. (Where the names of the worksheets are given by the dataNames property of the object.)

Works only when 'ext' input is set to 'xlsx'. (The default)

> 0 : The default saving method.

- 'strip' : > 'on' : Strip all observation dates where all the variables has no value. (is nan)

> 'off' : Default

Caution : Only an option when saved to excel.

- 'dateformat' : Sets the date format of the saved output Only if 'ext' are set to 'xlsx', which is default.
The supported date formats are:

> 'default'
> 'fame'
> 'xls'
> 'NBNorsk' ('NBNorwegian')
> 'NBEnglish' ('NBEngelsk')

See the toDates methods of the classes nb_day, nb_month, nb_quarter, nb_semiAnnual and nb_year for more information on the date formats

- 'sheets' : Set the names of the sheets to write to, as a cellstr. The length must match the number of pages of the nb_ts object. Only supported if 'ext' is set to 'xlsx'.

Output:

The nb_ts object saved to the wanted file format.

Examples:

Save the data to excel with the name 'test':
saveDataBase(obj,'saveName','test');

```
Save the data to excel with another date format:  
saveDataBase(obj,'saveName','test','dateformat','xls');  
  
Save the data to excel spreadsheet with the datasets as different  
worksheets:  
saveDataBase(obj,'saveName','test','append',1);  
  
Save the data to a .mat with the name 'test':  
obj.savDataBase('ext','mat','saveName','test')
```

See also:

[asCell](#), [toStructure](#), [nb_readMat](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

► **sec** ↑

obj = sec(obj)

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

obj = sec(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **secd** ↑

obj = secd(obj)

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = secd(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sech** ↑

```
obj = sech(obj)
```

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sech(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **set2nan** ↑

```
obj = set2nan(obj,testedVariable,settledVariable)
```

Input:

- obj : An object of class nb_ts
- testedVariable : A string with the name of the variable you are testing for nan values
- settedVariable : A string with the name of the variable you are setting equal to nan where the testedVariable are in fact nan.

Output:

- obj : An nb_ts object where the settedVariable are set equal to nan at all the same dates as the testedVariable is nan.

Examples:

```
obj = nb_ts([1,2;nan,1;2,3],'', '2012', {'Var1','Var2'});
```

```
obj = set2nan(obj,'Var1','Var2');
```

```
obj =
```

'Time'	'Var1'	'Var2'
'2012'	[1]	[2]
'2013'	[NaN]	[NaN]
'2014'	[2]	[3]

Written by Kenneth S. Paulsen

► **setDayOfWeek ↑**

```
obj = setDayOfWeek(obj,dayOfWeek)
```

Description:

Set the day the week represents when converted to daily frequency.

Input:

- obj : A nb_ts object representing weekly data.
- dayOfWeek : The day of the week. 1 is sunday, 2 is monday, ..., 7 is saturday.

Output:

- obj : A nb_ts object representing weekly data.

See also:

[nb_ts.convert](#)

► **setInfinite2NaN** ↑

```
obj = setInfinite2NaN(obj)
```

Description:

Set all elements that are infinite to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setMethodCalls** ↑

```
obj = setMethodCalls(obj,c)
```

Description:

Set all method calls of the object with a cell matrix. The object needs to be updatable.

Caution: To delete method calls use deleteMethodCalls

Caution: Call obj = update(obj) to interpret the changes made!

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.
- c : A cell matrix. See the output of getMethodCalls

Output:

- obj : An object of class nb_ts, nb_cs or nb_data.

See also:

[nb_ts.deleteMethodCalls](#), [nb_ts.getMethodCalls](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **setNewStartDate ↑**

`obj = setNewStartDate(obj,start)`

Description:

Set new start date of nb_ts object.

Input:

- `obj` : An object of class nb_ts.
- `start` : An object of class nb_date or a date string.

Output:

- `obj` : An object of class nb_ts with the new start date.

See also:

[nb_ts.lag](#), [nb_ts.lead](#)

Written by Kenneth Sæterhagen Paulsen

► **setToNaN ↑**

`obj = setToNaN(obj,startDateWin,endDateWin,variablesWin,Pages)`

Description:

Set a window of the object to nan

Input:

- `obj` : An object of class nb_ts
- `startDateWin` : The start date of the nan data of the object. Must be a string on the format; 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the start date of the data.

Can also be an object which is a subclass of the nb_date class.

- endDateWin : The end date of the nan data of the object. Must be a string on the format. 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the end date of the data.

Can also be an object which is a subclass of the nb_date class.

- variablesWin : A cellstr array of the variables you want to set to nan. If empty all the variables is set to nan.
- pages : A numerical index of the pages you want to set to nan.

Output:

- obj : An nb_ts object where the wanted data is set to nan.

See also:

[nb_ts](#)

Written by Kenneth S. Paulsen

► **setValue ↑**

```
obj = setValue(obj,VarName,Value,startDateOfValues, ...
endDateOfValues,pages)
```

Description:

Set some values of the dataset(s). (Only one variable at a time.) You cannot use dates outside the objects dates (i.e. outside obj.startDate:obj.endDate)

Input:

- obj : An object of class nb_ts
- VarName : The variable name of the variable you want to set some values of. As a string
- Value : The values you want to assign to the variable. Must be a numerical vector (with the same number of pages as given by pages or as the number of dataset the object consists of.) Cannot be outside the window of the data of the object.
- startDateOfValues : A date string with the start date of the data. If not given or given as a empty string it will assume that the setted data start at the startDate of the object.

This input could also be an object which is of a subclass of the nb_date class.

- endDateOfValues : A date string with the end date of the data. If not given or given as a empty string it will assume that the setted data end at the endDate of the object.

This input could also be an object which is of a subclass of the nb_date class

- pages : At which pages you want to set the values of a variable. Must be a numerical index of the pages you want to set.
E.g. if you want to set the values of the 3 first datasets (pages of the data) of the object. And the number of datasets of the object is larger then 3. You can use; 1:3. If empty all pages must be set.

Output:

- obj : An nb_ts object where the data of the given variable has been set.

Examples:

```
obj = nb_ts([2;3;4],'',,'2012',{'Var1'});  
  
obj = obj.setValue('Var1',[1;2;3]);  
obj = obj.setValue('Var1',2,'2013','2013',1);  
obj = obj.setValue('Var1',[1;2;3],'2012','2014');
```

See also
addVariable

Written by Kenneth S. Paulsen

► **setZero2NaN ↑**

```
obj = setZero2NaN(obj)
```

Description:

Set all elements that are 0 to nan.

Input:

- obj : An object of class nb_ts.

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► sgrowth ↑

```
obj = sgrowth(obj,horizon)
```

Description:

Calculates smoothed 12 or 6 months growth.

$x(*) = ((x13+x14+x15)/(x1+x2+x3)-1)*100$

$x(*) = ((x7+x8+x9)/(x1+x2+x3)-1)*100$

Note: Works only for monthly data.

Based on Anne Sofie Jores data transformation code.

Input:

- obj : A nb_ts object

- horizon : Either 1 or 2, depending on the frequency you want your growth to be calculated over. 1 is annual and 2 is semi-annual.

Output:

- out : A nb_ts object.

Examples:

```
a = nb_ts('forecast.db','','','{'NB.QUA_PCPIJAE'})  
a = sgrowth(a,1)
```

See also:

[nb_sgrowth](#)

Written by Tobias Ingebrigtsen

► shrinkSample ↑

```
obj = shrinkSample(obj,variables,periods)
```

Description:

Shrink sample to be balanced for a selected number of variables. (May tolerate trailing nan with the selected number of periods)

Input:

- obj : An object of class nb_ts
- variables : A cellstr with the variables of interest
- periods : A integer with the number of tolerate trailing nan. Default is 0.

Output:

- obj : An object of class nb_ts

See also:

[nb_ts.cutSample](#)

Written by Kenneth S. Paulsen

► **simulate ↑**

```
obj      = nb_ts.simulate()
obj      = nb_ts.simulate(start,obs,vars,pages,lambda,rho,dist,burn)
[obj,res] = nb_ts.simulate(start,obs,vars,pages,lambda,rho,dist,burn,...  
    varargin)
```

Description:

Simulate time-series data from a VAR-MA model

Input:

- start : The start date of the created nb_ts object.
- obs : The number of observation of the created nb_ts object.
- vars : Either a 1 x nVars cellstr with the variables of the nb_ts object, or a scalar with the number of variables of the returned nb_ts object.
- pages : Number of pages of the nb_ts object. I.e. the number of simulations from the process.
- lambda : The VAR coefficients. As a nVar x nVar*nLags double. Default is a zero matrix.
- rho : The MA coefficients (including the contemporaneous). As a nVar x nMA + 1. Default is a vector of ones. The weight on the contemporaneous innovation comes first, then the first

MA term and so on.

- dist : The distribution to draw the residuals from. As a 1 x svars nb_distribution object. Default is $N(0,1)$, i.e. when [] is given.
- burn : Number of observations used to simulate starting values. As an integer. Default is 10.

Optional input:

- 'exoData' : A obs * burn x nExo double with the simulated data of the exogenous variables.
- 'B' : A nVars x nExo double with the coefficients on the exogenous variables of the VAR.

Output:

- obj : A nb_ts object with the simulated data.
- res : A nb_ts object with the simulated innovations.

Examples:

```
% AR(2) model with std 0.5^2
obj = nb_ts.simulate('1994Q1',100,1,1000,[0.9,-0.2],0.5)

% A VAR with two variables and two lags
obj = nb_ts.simulate('1994Q1',100,2,1000, ...
    [0.9,-0.2,0.3,0;0.1,0,0.7,0.05],[0.5;0.3])
```

Written by Kenneth Sæterhagen Paulsen

► **sin ↑**

```
obj = sin(obj)
```

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = sin(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_DATASOURCE
```

► **sind** ↑

```
obj = sind(obj)
```

Description:

sind(obj) is the sine of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sind(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

sinh(obj) is the hyperbolic sine of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = sinh(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► **skewness ↑**

```
obj = skewness(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the skewness of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their skewness.

The output can also be set to be a double or a nb_cs object consisting of the skewness values of the data.

Input:

- obj : An object of class nb_ts
- flag : Same as for the function skewness created by MATLAB.
- outputType :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their skewness.
 - > 'double' : Get the skewness values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consist of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the skewness values as nb_cs object. The skewness will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the skewness over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method
- dimension : The dimension to calculate the skewness over.

Examples:

```
nb_tsObj = skewness(obj);
double    = skewness(obj,0,'double');
nb_csObj = skewness(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► **sort ↑**

```
obj = sort(obj,mode,dim,variable)
```

Description:

Sort the wanted dimension (observations) of the data in the order given by the order input.

Caution : Sorting does not reorder the dataNames, variables or dates!

Input:

- obj : An object of class nb_ts
- mode : 'descend' (default)
'ascend'
- dim : The dimension to sort. Either 1, 2 or 3. Default is 3.
- variable : Give a string with the variable to sort. The rest of the variables will then be reordered accordingly.
If empty (default), all variables are sorted.

Caution: Only an option for dim == 1

Output:

- obj : An object of class nb_ts

Examples:

```
in  = nb_ts(rand(10,2,10),'',1,{'Var1','Var2'});
out = sort(in);
out = sort(in,'descend')
```

Written by Kenneth S. Paulsen

► **sortProperty ↑**

```
obj = sortProperty(obj,type)
```

Description:

Sort the variables/types/datasets of the object alphabetically.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data
- type : Either 'variables' (default), 'types' or 'datasets'. 'types' is only supported for nb_cs objects.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **splitByPublishDates** ↑

```
out = splitByPublishDates(obj,publishDates,lag,format)
```

Description:

Input:

- obj : An object of class nb_ts with only one page!
- publishDates : A cellstr or nb_day vector with the dates that the series has been published backward in time.
- lag : The number of periods the series are when published. E.g. if the observation for the last month of a monthly series is published at the 10th of each month, then give 1 (default). Give 2 if it is published with lag of two months, and so on.
- format : The format of the publishing dates stored in the dataNames property of the output. See the nb_date.format2string method. Default is 'yyyymmdd'.

Output:

- out : An object of class nb_ts with pages given by the series up to the given date.

Written by Kenneth Sæterhagen Paulsen

► **splitSample** ↑

```
data = splitSample(obj,nSteps,appendNaN)
```

Description:

Split a sample into a multipaged nb_data object with size nSteps x nvar x nobs. Will append nan values for the last observations as default.

Input:

- data : An object of class nb_ts.
- nSteps : The number of period of the splitted data. As a scalar integer.
- appendNaN : true or false. Default is true.

Output:

- data : An object of class nb_data with size nSteps x nvar x nobs.

Examples:

```
data = splitSample(nb_ts.rand(1,20,2),4)
```

See also:

[nb_ts](#), [nb_ts.splitSeries](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **splitSeries ↑**

```
obj = splitSeries(obj,variable,date,postfix,overlapping)
```

Description:

Split a series into two variables at a given date.

Input:

- obj : An object of class nb_ts
- variable : The variable(s) to split. Either a string or a cellstr.
- date : The date to split the series. Either as a string or a nb_date object. Local variables syntax is supported.
- postfix : A string with the postfix of the new variable(s).
- overlapping : Indicate if you want the splitted series to be overlapping or not. As a logical. Default is true.

Output:

- obj : An object of class nb_ts with the splitted series added.

See also:

[nb_ts](#), [nb_ts.splitSample](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

sqrt(obj) is the square root of the elements of obj. Complex results are produced for non-positive elements.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

out = sqrt(in);

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **squeeze** ↑

obj = squeeze(obj)

Description:

Squeezes the different datasets (pages) of the given nb_ts object. It will append the variables in each dataset with the given data name

Input:

- obj : An nb_ts object. If the object has only one dataset (page) this method has nothing to do.

Output:

- obj : An nb_ts object, where all the pages are squeezed together. I.e. all data will be stored in one dataset, and all variable names will get the dataset name appended.

Example:

```
obj = nb_ts(ones(1,1,2),{'b','c'},'2012',{'Var1'})
```

```
obj =
```

```
(:,:,1) =
```

'Time'	'Var1'
'2012'	[1]

```
(:,:,2) =
```

'Time'	'Var1'
'2012'	[1]

```
sObj = squeeze(obj)
```

```
sObj =
```

'Time'	'Var1_b'	'Var1_c'
'2012'	[1]	[1]

Written by Kenneth Sæterhagen Paulsen

► **std ↑**

```
obj = std(obj,flag,outputType,dimension,varargin)
```

Description:

Calculate the std of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their std.

The output can also be set to be a double or a nb_cs object consisting of the std values of the data.

Input:

- obj : An object of class nb_ts
- flag : > 0 : normalises by N-1 (Default)
> 1 : normalises by N

Where N is the sample length.

- outputType :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their std.
 - > 'double' : Get the std values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the std values as nb_cs object. The std will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the std over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_tsObj = std(obj);
double   = std(obj,0,'double');
nb_csObj = std(obj,0,'nb_cs');
```

Written by Kenneth S. Paulsen

► **stdise** ↑

```
obj = stdise(obj,flag)
```

Description:

Standardise data of the object by subtracting mean and dividing by std deviation.

Input :

- obj : An object of class nb_ts
- flag : - 0 : normalises by N-1 (Default)

- 1 : normalises by N

Where N is the sample length.

Output:

- obj : An nb_ts object with the standardised data

Examples:

```
obj = stdise(obj);
```

Written by Kenneth S. Paulsen

► **stopSorting** ↑

```
obj = stopSorting(obj)
```

Description:

Stop sorting of variables.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data.

Output:

- obj : An object of class nb_ts, nb_cs or nb_data where sorting is turned off.

Example:

```
obj = obj.addPrefix('NEMO.');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **strip** ↑

```
obj = strip(obj,startDate,endDate,variables)
```

Description:

Sets all values of the given variables of a nb_ts object, between two given dates, to nan.

Input:

- obj : A nb_ts object
- startDate : The starting point of the strip, must be either an nb_date object or a valid input to the toDate method of the nb_date class
- endDate : The starting point of the strip, must be either an nb_date object or a valid input to the toDate method of the nb_date class
- variables : A cellstring with the variables to perform the method on. Can be empty, in which case the method will be performed on all variables in the object.

Output:

- obj : A nb_ts object

Examples:

```
strippedObj = strip(obj,'2011Q1','2012Q3',{'Var1','Var2'})  
strippedObj = strip(obj,'2011Q1','2012Q3')
```

See also:

[nb_ts.setToNaN](#)

Written by Eyo Herstad

► **stripButLast** ↑

```
obj = stripButLast(obj,numPeriods,variables)
```

Description:

Sets all values of the given variables of a nb_ts object, except a given amount of observations at the end, to nan.

Input:

- obj : A nb_ts object
- numPeriods : The amount of observations to be left as-is in the object. If the input is given to 2, the returned object will have all it's observations set to nan except the last two observations.
- variables : A cellstring with the variables to perform the method on. Can be empty, in which case the method will be performed on all variables in the object.

Output:

- obj : A nb_ts object

Examples:

```
strippedObj = stripButLast(obj,5,['Var1','Var2'])  
strippedObj = stripButLast(obj,'5')
```

Written by Eyo Herstad

► **stripFcstFromRealTime** ↑

```
obj = stripFcstFromRealTime(obj,type)
```

Description:

Remove conditional information from a real-time dataset

Caution : It is assumed that each page give rise to a new historical observation, i.e. the dataNames property must not increase by more than one period when it is traversed.

Input:

- obj : An object of class nb_ts. The dataNames property must either store the vintages or the end of history dates.
- type : Give 1 if the dataNames property stores vintages, otherwise use 2 if it stores the end of history dates. Default is 2:

Output:

- obj : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

```
s = struct(obj)
```

Description:

Object to struct.

Input:

- obj : An nb_ts object.

Output:

- s : A structure representing an nb_ts object.

Written by Kenneth Sæterhagen Paulsen

► structure2Dataset ↑

```
obj = structure2Dataset(obj,givenStruct,NameOfDataset,startDate)
```

Description:

When you give a struct to the constructor of this class all the fields af that struct will be added as a new dataset, but this method add all the fields of the input struct as a variable in one dataset

If you only want to add a dataset from a structure in this way (no other datasets) intitialize an empty object, and then use this method.

Caution : The structure must include a field 'startDate', which must be a string with the start date of the data to add.

Inputs:

- obj : An nb_ts object, could be empty.
- givenStruct : The structure which contains the data of the added variable. Where the fieldnames will be the variables name of the added dataset + plus one field 'startDate' with the start date of the dataset.
- NameOfDataset : Name of the added dataset. If not provided a default name will be given
- startDate : The start date of the added structure. Only needed if the provided structure has no field 'startDate'.

Outputs:

- obj : An nb_ts object with the added structure as a dataset.

Examples:

```
s          = struct();  
s.startDate = '2012';  
s.var1      = [2;2;2];  
data       = nb_ts();  
data       = data.structure2Dataset(s)  
  
data =
```

```
'Time'      'var1'  
'2012'      [    2]  
'2013'      [    2]  
'2014'      [    2]
```

Written by Kenneth S. Paulsen

► **subAvg** ↑

```
obj = subAvg(obj,k)  
obj = subSum(obj,k,w)
```

Description:

Will for the last k periods (including the present) calculate the cumulative sum and then divide by the amount of periods, which gives you the average over those periods.

Input:

- obj : An object of class nb_ts.
- k : Lets you choose what frequency you want to calculate the average over. As a double. E.g. 4 if you have quarterly data and want to calculate the average over the last 4 quarters.

Output:

- obj : An nb_ts object where the data are the average over the last k periods.

Examples:

```
dataquarter = nb_ts('histdata.db', '', '2000Q1', {'QSA_PCPIJAE'});  
growth12Q   = pcn(dataquarter,4);  
growth12Q   = subAvg(growth12Q,4);
```

See also:

[growth](#), [q2y](#), [pcn](#)

Written by Tobias Ingebrigtsen

► **subSum** ↑

```
obj = subSum(obj,k)  
obj = subSum(obj,k,w)
```

Description:

Calculates the cumulative sum over the last k periods (including the present period).

Input:

- obj : An object of class nb_ts.
- k : Lets you choose what frequency you want to calculate the cumulative sum over. As a double. E.g. 4 if you have quarterly data and want to calculate the average over the last 4 quarters.
- w : Weights applies to the k periods to sum over. Default is equal weights! As a double vector with length k.

Output:

- obj : An object of class nb_ts.

Examples:

```
dataquarter = nb_ts('histdata.db', '', '2000Q1', {'QSA_PCPIJAE'});  
growth12Q = pcn(dataquarter,4);  
growth12Q = subSum(growth12Q,4);
```

See also:

[pcn](#), [growth](#), [subAvg](#)

Written by Tobias Ingebrigtsen

► subsasgn ↑

```
varargout = subsasgn(obj,s,b)
```

Description:

Makes it possible to do subscripted assignment of the data of an object

Input:

- obj : An object of class nb_ts
- s : The same input as when you do subscripted assignment one a double matrix
- b : The assign data at the given index s.

Output:

- obj : The assign data property setted of the given object.

Examples:

```
obj      = nb_ts([2;3;4],'', '2012', {'Var1'});
obj(1:3,1) = [1;2;3]
obj =
    'Time'    'Var1'
    '2012'    [ 1]
    '2013'    [ 2]
    '2014'    [ 3]
```

Written by Kenneth S. Paulsen

► subsref ↑

```
varargout = subsref(obj,s)
```

Description:

Makes it possible to do subscripted reference of the data of an object. See the examples for more.

Using the method window, just using indexing instead.

Input:

- obj : An object of class nb_ts.
- s : A structure on how to index the object.

Output:

- obj : The indexed object.

Examples:

```
obj = obj(1,2:4,1)
```

same as

```
obj = obj(1,2:4)
```

Be aware that that obj(1,:) and obj(1,2:end) will also work

```
obj = obj('1994Q1')
```

same as

```

obj.window('1994Q1','1994Q1')

obj = obj('1994Q1','1994Q4')

same as

obj.window('1994Q1','1994Q4')

obj = obj('1994Q1','1994Q4',{'Var1','Var2',...})

same as

obj.window('1994Q1','1994Q4',{'Var1','Var2',...})

obj = obj('1994Q1','1994Q4',{'Var1','Var2',...},1:2)

same as

obj.window('1994Q1','1994Q4',{'Var1','Var2',...},1:2)

obj = obj('Var1','Var2',...)

same as

obj.window('','','',{'Var1','Var2',...},1:obj.numberOfDatasets)

```

Written by Kenneth S. Paulsen

► sum ↑

```
obj = sum(obj,dim)
```

Description:

Takes the sum of the object timeseries

Caution : All sums ignore nan values. If not all values is nan.

Input:

- obj : An object of class nb_ts
- dim : The dimension to sum over, returns:
 - > dim = 1: Either :
 - > An nb_ts object with all the elements of each variable set to their sum over the time horizon.

```

> An nb_cs with one type representing the sum
over the time horizon). The type is named
'sum'.

> dim = 2: An nb_ts object with one variable representing
the sum (Take the sum over the variables)
(Default)

> dim = 3: An nb_ts object with only on page, representing
the sum over all pages.

- output : Either 'nb_ts' (default) or 'nb_cs'. Only important
when summing over the 1 dimension.

```

Output:

- obj : Either an nb_ts object or an nb_cs object representing
the sum.

Examples:

```

obj      = nb_ts([2,1;3,2;4,3],'', '2012', {'Var1','var2'})
obj =
    'Time'      'Var1'      'var2'
    '2012'      [ 2]      [ 1]
    '2013'      [ 3]      [ 2]
    '2014'      [ 4]      [ 3]

s1 = sum(obj,1,'nb_ts')

s1 =
    'Time'      'Var1'      'var2'
    '2012'      [ 9]      [ 6]
    '2013'      [ 9]      [ 6]
    '2014'      [ 9]      [ 6]

s1_cs = sum(obj,1,'nb_cs')

s1_cs =
    'Types'      'Var1'      'var2'
    'sum'        [ 9]      [ 6]

s2 = sum(obj,2)

s2 =
    'Time'      'sum'
    '2012'      [ 3]
    '2013'      [ 5]
    '2014'      [ 7]

s2 = sum(obj,3)

s2 =

```

```
'Time'      'Var1'      'var2'  
'2012'      [    2]      [    1]  
'2013'      [    3]      [    2]  
'2014'      [    4]      [    3]
```

Written by Kenneth S. Paulsen

► **sumOAF** ↑

```
obj = sumOAF(obj,sumFreq)
```

Description:

Take the sum over a lower frequency.

I.e. if frequency is quartely, this function can sum over all the quarters of a year. And it will return the sum over the year in all quarters of the that year

Input:

- obj : An object of class nb_ts
- sumFreq : The frequency to sum over

Output:

- obj : An object of class nb_ts where the data is summed over a lower frequency

Examples:

```
obj = nb_ts.ones('2012Q1',8,1);  
obj = obj.sumOAF(1)
```

Written by Kenneth S. Paulsen

► **summary** ↑

```
obj = summary(obj)
```

Description:

Summary of the timesseries of the nb_ts object

Input:

- obj : An object of class nb_ts

Output:

- obj : The summary statistics - min, max, std, var, skewness and kurtosis, as a nb_cs object

Written by Kenneth S. Paulsen

► **tail** ↑

`tail(obj,nRows,page)`

Description:

Display the last n rows of a nb_ts object.

Input:

- obj : An nb_ts object.
- nRows : An integer with the number of rows to display. Defaults to 6.
- page : An integer with what page to display. Defaults to 1.

See also:

[nb_ts](#), [window](#), [head](#)

Written by Per Bjarne Bye

► **tan** ↑

`obj = tan(obj)`

Description:

Take the tangent of the data stored in the nb_ts, nb_cs or nb_data object

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
obj = tan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **tand** ↑

```
obj = tand(obj)
```

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tand(in);
```

Written by Andreas Haga Raavand

Inherited from superclass NB_DATASOURCE

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

tanh(obj) is the hyperbolic tangent of the elements of obj.

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An object of class nb_ts, nb_cs or nb_data

Examples:

```
out = tanh(in);  
  
Written by Andreas Haga Raavand  
  
Inherited from superclass NB_DATASOURCE
```

► **times** ↑

```
obj = times(obj,DB)
```

Description:

Element-wise multiplication (.*)

Caution : Will only multiply the data of corresponding variables of the two objects. I.e. variable not found to be in both object will result in timeseries with all nan values.

Input:

- obj : An object of class nb_ts
- DB : An object of class nb_ts

Output:

- obj : An nb_ts object the data results from element-wise multiplication of the data of the input objects.

Examples:

```
obj = obj.*DB;
```

See also:

[nb_ts.mtimes](#), [nb_ts.callop](#), [nb_ts.callfun](#)

Written by Kenneth S. Paulsen

► **timesCS** ↑

```
obj = timesCS(obj,DB)
```

Description:

Element-wise multiplication (.*) between a nb_ts and nb_cs object.

Caution : Will multiply the data of variables in obj that match does one variable in DB. The variable that does not match are not changed. The variables found in DB, but not in obj are ignored. No warning will be given if no matches are found.

Two cases:

> obj has as many pages as DB, and the number of pages are > 1, then the multiplication will be done page by page. DB can only have one type (row) in this case.

> DB has only one page, but has more than one type (row). Then obj must either have one page or as many pages as there are types of DB. If obj has one page th multiplication will be done for each type (row) of DB. The output will have as many pages as there are types of the DB input. If obj has the same number of types as DB, then each page of obj will be multiplied with the matching type of DB, i.e. page one is multiplied by type one and so on.

Input:

- obj : An object of class nb_ts.
- DB : An object of class nb_cs, can only have one type.

Output:

- obj : A nb_ts object the data results from element-wise multiplication of the data of the input objects.

Examples:

obj = obj.*DB;

See also:

[nb_ts.times](#)

Written by Kenneth S. Paulsen

► **toMFile ↑**

```
string = toMFile(obj)
toMFile(obj,filename)
string = toMFile(obj,filename,varName)
```

Description:

Write a .m file to generate the current object.

Input:

- obj : An nb_ts object.
- filename : The filename to write the code to. If empty no file will be written.
- varName : Name of decleared variable by this code. Default is data.

Output:

- string : A cellstr with the .m code to generate the current object.

Written by Kenneth Sæterhagen Paulsen

► **toRise_ts ↑**

```
data_ts = toRise_ts(obj)
data_ts = toRise_ts(obj,'struct')
```

Description:

Transform from an nb_ts object to an RISE ts object, or a struct of rise ts object representing each variable (Where the pages are added as dimension 2. I.e. the input to the data property of the rise_generic object).

Input:

- obj : An object of class nb_ts
- type : Either 'struct' or 'object'.

Output:

- data_ts : An object of class ts or a struct of ts objects

Examples:

```
ts = obj.toRise_ts();
```

Written by Kenneth S. Paulsen

► **toStructure ↑**

```
retStruct = toStructure(obj)
retStruct = toStructure(obj,old)
```

Description:

Transform an nb_ts obj to a structure

Input:

- obj : An object of class nb_ts
- old : Indicate if old mat file format is wanted. Default is 0 (false).

Output:

- retStruct : A structure with fieldsnames given by the object variables and field given by the variables data. (Can have more than one page). And a own field with the start date of the data. (As a string.)

Examples:

```
obj = nb_ts(ones(8,1),'', '2012Q1', {'Var1'});  
s = obj.toStructure()  
  
s =  
  
    Var1: [8x1 double]  
variables: {'Var1'}  
startDate: '2012Q1'  
    class: 'nb_ts'  
  userData: ''  
localVariables: []  
      sorted: 1
```

See also:

[nb_ts.struct](#)

Written by Kenneth S. Paulsen

► **toTseries ↑**

tseries_DB = toTseries(obj)

Description:

Transform from an nb_ts object to an tseries object

To use this function you need to add the IRIS package to the MATLAB path.

Input:

- obj : An nb_ts object

Output:

- tseries_DB : An tseries object

Examples:

```
tseries_DB = obj.toTseries();
```

Written by Kenneth S. Paulsen

► **todyn_ts** ↑

```
dyn_ts_DB = todyn_ts(obj)
```

Description:

Transform from an nb_ts object to an dyn_ts object

Input:

- obj : An object of class nb_ts

Output:

- dyn_ts_DB : An object of class dyn_ts

Examples:

```
dyn_ts_DB = obj.todyn_ts();
```

Written by Kenneth S. Paulsen

► **tonb_cell** ↑

```
nb_cell_DB = tonb_cell(obj,dateFormat)
```

Description:

Transform from a nb_ts object to a nb_cell object

Input:

- obj : An object of class nb_ts
- dateFormat : The date format of the dates of the return nb_cell object.
 - > 'default' (default)
 - > 'fame'
 - > 'NBEnglish' or 'NBEngelsk'
 - > 'NBNorsk' or 'NBNorwegian'
 - > 'xls'
- strip : - 'on' : Strip all observation dates where all the variables has no value.
- 'off' : Does not strip all observation dates where all the variables has no value. Default.

Output:

- nb_cell_DB : An object of class nb_cell

Examples:

```
nb_cell_DB = obj.tonb_cell();
```

Written by Kenneth S. Paulsen

► **tonb_cs ↑**

```
nb_cs_DB = tonb_cs(obj,dateFormat)
```

Description:

Transform from an nb_ts object to an nb_cs object

Input:

- obj : An object of class nb_ts
- dateFormat : The date format of the dates of the return nb_cs object.
 - > 'default' (default)
 - > 'fame'
 - > 'NBEnglish' or 'NBEngelsk'
 - > 'NBNorsk' or 'NBNorwegian'
 - > 'xls'
- strip : - 'on' : Strip all observation dates where all the variables has no value.
- 'off' : Does not strip all observation dates where all the variables has no value. Default.

Output:

- nb_cs_DB : An object of class nb_cs

Examples:

```
nb_cs_DB = obj.tonb_cs();
```

Written by Kenneth S. Paulsen

► **tonb_data** ↑

```
nb_data_DB = tonb_data(obj)
nb_data_DB = tonb_data(obj,start)
```

Description:

Transform from a nb_ts object to a nb_data object. The startObs will automatically be set to 1, if not the start input is provided.

Input:

- obj : An object of class nb_ts
- start : Start observation.

Output:

- nb_data_DB : An object of class nb_data

Examples:

```
nb_data_DB = obj.tonb_data();
```

Written by Kenneth S. Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : A nb_ts, nb_cs or nb_data object where all the data are the unary minus of the input objects data

Examples:

```
obj = -obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► undiff ↑

```
obj = undiff(obj,initialValues,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series diff. Inverse of the diff method.

Input:

- obj : An object of class nb_ts
- initialValues : A nb_ts object with a bigger window than obj, or a scalar or a double with the initial values of the indicies. Must be of the same size as the number of variables of the nb_ts object. If not provided 100 is default.

Caution : If periods > 1 the initialValues input must be given, and has at least have periods number of rows.
- periods : The number of periods the initial series has been taken diff over.

Output:

- Output : An nb_ts object with the indicies.

Examples:

```
obj = undiff(obj);
obj = undiff(obj,100);
obj = undiff(obj,[78,89]);
```

See also
nb_ts.diff

Written by Kenneth S. Paulsen

► **unstruct** ↑

```
obj = nb_ts.unstruct(s)
```

Description:

Reverse of the struct method.

Input:

- s : See the s output of the struct method.

Output:

- obj : An nb_ts object.

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,warningOff,inGUI)
```

Description:

This method will try to update the data of the nb_ts object.

Caution : It is only possible to update an nb_dataSource object which has updateable links to a FAME database or a full directory (path) to an excel spreadsheet or .mat file or the SMART database.

Caution : If you want to create a link to a specific worksheet of a excel file you must provide the extension!

Caution : All method calls done on the object are preserved.
(There exist some exception; and, or etc.)

Input:

- obj : An object of class nb_dataSource which are updatable.

- warningOff : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'.

- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.

Output:

- obj : An object of class nb_dataSource with the data updated. If it is not possible a warning will be given.

Examples:

```
obj = nb_ts(['N:\Matlab\Utvikling\MATLAB_LIB\NEMO\'...
             'Documentation\Examples\example_ts_quarterly']);
obj = obj.update
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_DATASOURCE

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus

Input:

- obj : An object of class nb_ts, nb_cs or nb_data

Output:

- obj : An nb_ts, nb_cs or nb_data object where all the data are the unary plus of the input objects data

Examples:

```
obj = +obj;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_DATASOURCE

► **var** ↑

```
obj = var(obj,outputType,dimension,varargin)
```

Description:

Calculate the var of each timeseries. The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their var.

The output can also be set to be a double or a nb_cs object consisting of the var values of the data.

Input:

- obj : An object of class nb_ts
- outputType :
 - > 'nb_ts' : The result will be an object of class nb_ts where all the non-nan values of all the timeseries are being set to their var.
 - > 'double' : Get the var values as doubles, where each column of the double matches the variable location in the 'variables' property. If the object consists of more pages the double will also consist of more pages.
 - > 'nb_cs' : Get the var values as nb_cs object. The variance will when this option is used only be calculated over the number of observations. (I.e. dimension set to 1.)
- dimension : The dimension to calculate the var over.

Optional input:

- 'notHandleNaN' : Give this to not handle nan values.

Output:

- obj : See the input outputType for more one the output from this method

Examples:

```
nb_tsObj = var(obj);
double   = var(obj,'double');
nb_csObj = var(obj,'nb_cs');
```

Written by Kenneth S. Paulsen

► vertcat ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_ts
- b : An object of class nb_ts
- varargin : Optional numbers of objects of class nb_ts

Output:

- a : An nb_ts object where the data from the different objects are appended to each other.

Examples:

```
obj = nb_ts(ones(2,1),'', '2012Q1', {'Var1'});  
aObj = nb_ts(ones(2,1),'', '2012Q3', {'Var1'});  
m = [obj;aObj]
```

m =

```
'Time'      'Var1'  
'2012Q1'    [ 1]  
'2012Q2'    [ 1]  
'2012Q3'    [ 1]  
'2012Q4'    [ 1]
```

Written by Kenneth S. Paulsen

► whiten ↑

```
F = whiten(obj)  
F = whiten(obj,r)
```

Description:

Calculate of factors from X that have variance one and covariances 0.

Input:

- obj : An object of class nb_ts with size nObs x nVar x nPage.
- r : The number of principal component. If empty nVar factors are returned.

Output:

- F : The principal component, as a nb_ts object.

Example:

```
load hald;  
ingredients = nb_ts(ingredients,'',1,{'tricalcium_aluminate',...  
                                'tricalcium_silicate',...  
                                'tetracalcium_aluminoferrite',...  
                                'beta_dicalcium_silicate'});  
F = whiten(ingredients)
```

See also:

[nb_whiten](#), [nb_ts.pca](#)

► **window** ↑

```
obj = window(obj,startDateWin,endDateWin,variablesWin,pages)
```

Description:

Narrow down window of the data of the nb_ts object

Input:

- obj : An object of class nb_ts
- startDateWin : The new wanted start date of the data of the object. Must be a string on the format; 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the start date of the data.
Can also be an object which is a subclass of the nb_date class.
- endDateWin : The new wanted end date of the data of the object. Must be a string on the format. 'yyyyQq', 'yyyyMmm' or 'yyyy'. If empty, i.e. '', nothing is done to the end date of the data.
Can also be an object which is a subclass of the nb_date class.
- variablesWin : A cellstr array of the variables you want to keep of the given object. If empty all the variables is kept.
- pages : A numerical index of the pages you want to keep. E.g. if you want to keep the first 3 datasets (pages of the data) of the object. And the number of datasets of the object is larger than 3. You can use; 1:3. If empty all pages is kept.
I.e. obj = obj.window('',{},1:3)

Output:

- obj : An nb_ts object where the datasets of the object is narrowed down to the specified window.

Examples:

```
obj = nb_ts(ones(10,2,2),'', '2012Q1', {'Var1','Var2'});  
obj = obj.window('2012Q2')
```

```
obj = obj.window('','2013Q4')
obj = obj.window('','','',{'Var1'})
obj = obj.window('','','{}',1)
```

See also:

Written by Kenneth S. Paulsen

► **writeTex** ↑

```
writeTex(obj,filename)
writeTex(obj,filename,'inputName', inputValue,...)
```

Description:

Writes the object data to a LaTeX table saved to a .tex file.

If the object consists of more pages (datasets) more tables will be created.

Input:

- obj : An object of class nb_ts
- filename : The name of the saved .tex file. With or without the extension.

Optional input:

See optional inputs to nb_ts.getCode.

Output:

A saved .tex file with the LaTeX table

Examples:

```
obj = nb_ts(rand(10,3),'','2012Q1',{'Var1','Var2','Var3'});
obj.writeTex('test')
```

See also:

[nb_ts](#), [nb_ts.getCode](#)

Written by Kenneth Sæterhagen Paulsen

► **x12Census** ↑

```
obj = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj,varargin)
```

Description:

Do x12 Census adjustement to an nb_ts object using x12awin.exe.

This is a file copied from the IRIS Toolbox an adapted to the NB Toolbox. (I.e. using nb_ts instead of tseries)

x12awin.exe is a program developed by:

U. S. Department of Commerce, U. S. Census Bureau

X-12-ARIMA quarterly seasonal adjustment Method,
Release Version 0.3 Build 192

This method modifies the X-11 variant of Census Method II
by J. Shiskin A.H. Young and J.C. Musgrave of February, 1967.
and the X-11-ARIMA program based on the methodological research
developed by Estela Bee Dagum, Chief of the Seasonal Adjustment
and Time Series Staff of Statistics Canada, September, 1979.

This version of X-12-ARIMA includes an automatic ARIMA model
selection procedure based largely on the procedure of Gomez and
Maravall (1998) as implemented in TRAMO (1996).

Caution : The object must be of quarterly or monthly frequency!!

Input:

- obj : Input data that will seasonally adjusted or filtered
by the Census X12 Arima. Must be an nb_ts object.

Optional input (..., 'propertyName', PropertyValue, ...):

- 'backcast' : Run a backcast based on the fitted ARIMA model
for this number of periods back to improve on the
seasonal adjustment. The backcast is included
in the output argument x. Must be a scalar.

- 'log' : Logarithmise the input data before, and
de-logarithmise the output data back after,
running x12. {1} | 0

- 'forecast' : Run a forecast based on the fitted ARIMA model
for this number of periods ahead to improve on
the seasonal adjustment. The forecast is included
in the output argument x. Must be a scalar.

- 'display' : Display X12 output messages in command window.
{0} | 1. Default is not.

- 'dummy' : Dummy variable or variables (in case of a
multivariate nb_ts object) used in X12-ARIMA
regression; the dummy variables can also include
values for forecasts and backcasts if you request

them. Either an nb_ts object or empty.

- 'dummyType' : Type of dummy. Either 'ao' | {'holiday'} | 'td'
- 'mode' : Seasonal adjustment mode. 'auto' means that series with only positive or only negative numbers will be adjusted in the 'mult' (multiplicative) mode, while series with combined positive and negative numbers in the 'add' (additive) mode. {'auto'} | 'add' | 'logadd' | 'mult' | 'pseudoadd' | 'sign'
- 'maxIter' : Maximum number of iterations for the X12 estimation procedure. Must be numeric. Default is 1500.
- 'maxOrder' : A 1-by-2 vector with maximum order for the regular ARMA model (can be 1, 2, 3, or 4) and maximum order for the seasonal ARMA model (can be 1 or 2). Must be numeric. Default is [1,2].
- 'missing' : Allow for in-sample missing observations, and fill in values predicted by an estimated ARIMA process; if false, the seasonal adjustment will not run and a warning will be thrown. 1 | {0}.
- 'output' : Requested output data, as a string. Either:
 - 'irregular' : For the irregular component,
 - 'seasadj' : For the final seasonally adjusted series. Default.
 - 'seasonal' : For seasonal factors.
 - 'trendcycle' : For the trend-cycle.
 - 'missingvaladj' : For the original series with missing observations replaced with ARIMA estimates.
- 'specFile' : Name of the X12-ARIMA spec file. Must be a char. Default is 'default'.
- 'tdays' : Correct for the number of trading days. 1 | {0}
- 'tolerance' : Convergence tolerance for the X12 estimation procedure. Must be numeric. Default is 1e-5.
- 'startDate' : Start date of seasonally adjustment.
- 'endDate' : End date of seasonally adjustment.

Output:

- obj : An nb_ts object with the seasonally/trend adjusted/component of all the stored series of the object.

- **x** : The original object with the backcast and forecast appended.

Caution : If the object is link to a data source this object will still not be updateable.

- **outputfile** : A cellstr array with the output files from the seasonal/trend adjustment.
- **errorfile** : A cellstr array with error files from the seasonal/trend adjustment.
- **mdl** : A struct with ARIMA model estimates.

Written by Kenneth SÃ¶terhagen Paulsen

► **zeros** ↑

```
obj = nb_ts.zeros()
obj = nb_ts.zeros(start,obs,vars,pages,sorted)
```

Description:

Create a nb_ts object with all data set to 0.

Input:

- **start** : The start date of the created nb_ts object.
- **obs** : The number of observation of the created nb_ts object.
- **vars** : Either a cellstr with the variables of the nb_ts object, or a scalar with the number of variables of the nb_ts object.
- **pages** : Number of pages of the nb_ts object.
- **sorted** : true or false. Default is true.

Output:

- **obj** : An nb_ts object.

Examples:

```
obj = nb_ts.zeros('1',10,{'Var1','Var2'});
obj = nb_ts.zeros('2012Q1',2,2);
obj = nb_ts.zeros('2012Q1',2,2,10);
```

See also:

[nb_ts](#)

◆ nb_cell2obj

```
data = nb_cell2obj(cellMatrix)
```

Description:

This is a function tries to transform an cell matrix to an nb_data, nb_ts or nb_cs object.

Input:

- cellMatrix : A cell matrix. Must have size(1) > 1 and size(2) > 1
- excel : true or false. Default is false.
- sorted : true or false. Default is false.

Formats:

- Timeseries :

```
{'dates',      'Var1', 'Var2';  
'30.06.2012', 3,    4;  
'30.09.2012', 5,    6}
```

- Cross sectional data :

```
{'types', 'Var1', 'Var2';  
'var',   1,     4;  
'std',   1,     2}
```

- Dimensionless (Indicated by having 'obs'/'observations' in the element {1,1} of the cell matrix)

```
{'obs',      'Var1', 'Var2';  
1,          1,     4;  
2,          1,     2}
```

- Business days / timeseries with missing observations (Indicated by having 'bd'/'businessdays'/'md'/'missingdates' in the element {1,1} of the cell matrix)

```
{'md',      'Var1', 'Var2';  
1,          1,     4;  
2,          1,     2}
```

- excel : If the cell comes from a excel spreadsheet, sometimes the data needs to be stripped.

Output:

- data : An object of class nb_data, nb_ts or nb_cs

See also:

[nb_ts](#), [nb_cs](#), [nb_data](#)

Written by Kenneth S. Paulsen

◆ **[nb_checkPostFix](#)**

message = nb_checkPostFix(postfix)

Description:

Check the postfix for unsupported chars

Written by Kenneth Sætherhagen Paulsen

◆ **[nb_checkPreFix](#)**

message = nb_checkPreFix(postfix)

Description:

Check the prefix for unsupported chars

Written by Kenneth Sætherhagen Paulsen

◆ **[nb_covidDates](#)**

[dates,drop] = nb_covidDates(freq)

Description:

Get the dates of the covid episode.

Input:

- freq : The frequency of the covid episodes.

Output:

- dates : A 1 x nDates nb_date array.

- drop : Number of dummies set to all 0, i.e. the X number of dummies at the end.

See also:

[nb_ts.covidDummy](#), [nb_covidDummyNames](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_covidDummyNames**

```
names = nb_covidDummyNames(freq)
names = nb_covidDummyNames(freq, covidDummyDates)
```

Description:

The dummy names for the covid-19 episode.

Input:

- freq : The frequency of the covid dummy.
- covidDummyDates : A cellstr with the dates of the covid-19 crisis.
Default is the period 16.03.2020 – 31.12.2021.
Hint: Use nb_month(3,2020):nb_month(12,2021).

Output:

- names : A cellstr with the names of the covid dummies.

See also:

[nb_ts.covidDummy](#), [nb_covidDates](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_createDefaultLink**

```
newLink = nb_createDefaultLink()
```

Description:

Creates a default link (i.e. empty) used by the nb_ts and nb_cs classes

Output:

- newLink : A structure with all the fields of an link.

See also:

[nb_ts](#), [nb_cs](#), [nb_data](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_deleteDefaultWorksheets

```
nb_deleteDefaultWorksheets(filename)
```

Description:

Delete the 'Sheet1','Sheet2','Sheet3' worksheets of an excel spreadsheet

Input:

- filename : The excel filename, as a string.

Written by Kenneth Sæterhagen Paulsen

◆ nb_eval

```
out = nb_eval(expression,variables,data)
out = nb_eval(expression,variables,data,macro)
```

Description:

Evaluate a expression as a string.

Input:

- expression : A string with the expression to evaluate. E.g.
 'lag(exp(Var1/(Var2^1^3))*4,1)*Var4 + Var1*(exp(Var1/2))'
- variables : The variables of the dataset. As a cellstr with size
 1 x nvars.
- data : The dataset. A double or a nb_math_ts object with the
 data of the variables. Must have size nobs x nvars.
- macro : Set to true to also handle the NB Toolbox macro processing
 language as well. Default is false.

Output:

- out : The result of the evaluation of the string

See also:

[nb_ts.createVariable](#), [nb_cs.createVariable](#), [nb_cs.createType](#)
[nb_data.createVariable](#), [eval](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_excelCellOffset

```
xlcell = nb_excelCellOffset(base,x,y)
```

Description:

Get the excel cell index the number of increments up/down and to the left/right of the cell given by start.

Input:

- base : A string with the base cell. E.g. 'A1'
- x : A number of increments up (negative), or down (positive).
- y : A number of increments to the left (negative), or to the right (positive).

Output:

- xlcell : A string on the format 'A1'.

See also:

[nb_excelRange](#), [nb_letter2num](#), [nb_num2letter](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_excelRange**

```
range = nb_excelRange(base,r,c,br,bc)
```

Description:

Get excel range from a base cell on the format 'A1' and different combinations of offsets.

Input:

- base : A String on the format 'A1', 'ZZ123' or 'A1:B1'.
- If nargin == 2:
 - r : The range will be returned from the base with this number of row offsets. As a strictly positiv scalar integer.
 - c : The range will be returned from the base with this number of column offsets. As a strictly positiv scalar integer.
- If nargin == 4:
 - r : The range will be returned from the base with this number of row offsets. As a strictly positive scalar integer. 1 imply no movement.

- c : The range will be returned from the base with this number of column offsets. As a strictly positive scalar integer. 1 imply no movement.
- br : First move the base these number of rows. As a scalar integer. 0 imply no movement.
- bc : First move the base these number of columns. As a scalar integer. 0 imply no movement.

Output:

- range : A excel range on the format 'A1:C2' or 'B1:C2'.

Examples:

```
nb_excelRange('A1',2,2)
nb_excelRange('A1',2,2,1,1)
```

See also:

[nb_excelCellOffset](#), [nb_letter2num](#), [nb_num2letter](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_findIndex

```
index = nb_findIndex(cellstr1,cellstr2)
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_input2String

```
string = nb_input2String(anyObject)
```

Description:

Converts a input into a string. This function is used by the method `getMethodCalls` of the classes `nb_ts`, `nb_cs` and `nb_data`.

Caution : Some inputs that are classified as not editable will be returned as a string, i.e. 'Not editable'.

Input:

- anyObject : Any type of object

Output:

- string : A string

Written by Kenneth Sæterhagen Paulsen

◆ nb_isQorM

```
[ind,frequency] = nb_isQorM(date)
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_letter2num

```
num = nb_letter2num(letter)
```

Description:

Convert from 'A', 'AA', 'AZZ' to a number. The letter must be in the set ['A', 'Z'].

Input:

- letter : A one line char with length from 1-3.

Output:

- num : A number representing the letter combination.

Examples:

```
num1 = nb_letter2num('A')
num2 = nb_letter2num('Z')
num3 = nb_letter2num('AA')
num4 = nb_letter2num('AZ')
num5 = nb_letter2num('BA')
num6 = nb_letter2num('AZZ')
num7 = nb_letter2num('BAA')
num8 = nb_letter2num('ZZZ')
```

See also:

[nb_num2letter](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_lookUpVariables

```
[vars,numberOfLines] = nb_lookUpVariables(variables,lookUpMatrix,language,numberOfLinesOutput)
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_mergeSources

```
[sources,vintages,variables ] = ...
    nb_mergeSources(dbNames,vintages,variables)
```

Description:

A function to avoid duplicate sources in a dataset.

Input:

- dbNames : The names of the databases of the object.
- vintages : The vintages of the different variables of the object.
- variables : The variablenames of the object.

Output:

- sources : The databases, without duplicates
- vintages : The vintages returned without duplicates
- variables : The variables of the object with correct sources.

See also:

[nb_ts.merge](#)

Original code by Kenneth Sætherhagen Paulsen, tweaks by Eyo I. Herstad

◆ **nb_num2letter**

```
letter = nb_num2letter(num)
```

Description:

Convert a number to a letter. Must be a postive integer in [1,18278].

Input:

- num : A number representing the letter combination.

Output:

- letter : A one line char with length from 1-3.

Examples:

```
11 = nb_num2letter(nb_letter2num('A'))
12 = nb_num2letter(nb_letter2num('Z'))
13 = nb_num2letter(nb_letter2num('AA'))
14 = nb_num2letter(nb_letter2num('AZ'))
15 = nb_num2letter(nb_letter2num('BA'))
16 = nb_num2letter(nb_letter2num('AZZ'))
17 = nb_num2letter(nb_letter2num('BAA'))
18 = nb_num2letter(nb_letter2num('ZZZ'))
```

See also:

[nb_num2letter](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_readExcel**

```
data = nb_readExcel(filename)
data = nb_readExcel(filename,sheet)
data = nb_readExcel(filename,sheet,range)
data = nb_readExcel(filename,sheet,range,transpose)
data = nb_readExcel(filename,sheet,range,transpose,sorted)
```

Description:

This is a function for reading an excel spreadsheet and return, a nb_data object (dimensionless data), a nb_ts object (timeseries data) or a nb_cs object (cross sectional data).

Input:

- filename : A string with the filename of the excel spreadsheet.
If full path is given the data will be updateable.
- sheet : A string with the sheet name you want to read.
- range : A cell with the range of the dates/types/obs,
variables and data. (1 x 3 cell array).
E.g. {'A2:A5','B1:C1','B2:C5'}
- transpose : 1 (true) or 0 (false). Default is 0.
- sorted : 1 (true) or 0 (false). Default is 1.
- c : A cell with already read data from excel

Formats:

- Dimensionless:

obs	Var1	Var2
1	3	4
2	5	6

Caution : obs in the cell A1:A1 is required!!!

Caution : Nothing else could be included in the worksheet as
long as the range is not specified!!

- Timeseries :

dates	Var1	Var2
30.06.2012	3	4
30.09.2012	5	6

```
Caution : Nothing else could be included in the worksheet as
long as the range is not specified!!
```

- Cross sectional data :

```
types Var1  Var2
var      1      4
std      1      2
```

```
Caution : Nothing else could be included in the worksheet as
long as the range is not specified!!
```

Output:

- data : An object of class nb_data, nb_ts or nb_cs

See also:

[nb_data](#), [nb_ts](#), [nb_cs](#)

Written by Kenneth S. Paulsen

◆ **nb_readExcelMorePages**

```
data = nb_readExcelMorePages(filename)
data = nb_readExcelMorePages(filename, sorted, sheets)
```

Description:

This is a function for reading each sheet of an excel spreadsheet and return, a nb_data object (dimensionless data), a nb_ts object (timeseries data) or a nb_cs object (cross sectional data).

This function can then be utilized to read real-time or panel data from excel spreadsheets.

If you want to read the data as real-time, each sheet must be given the name of the end date of that vintage series. The format of the date must be supported by NB toolbox. Each vintage must give rise to one new observation (and only one!).

Input:

- filename : A string with the filename of the excel spreadsheet.
If full path is given the data will be updateable.
- sorted : 1 (true) or 0 (false). Default is 1.
- sheets : A cellstr with the sheet names you want to read. Default is all.

Formats (of each sheet):

- Dimensionless:

```
obs      Var1 Var2
1        3     4
2        5     6
```

Caution : obs in the cell A1:A1 is required!!!

Caution : Nothing else could be included in the worksheet as long as the range is not specified!!

- Timeseries :

```
dates      Var1 Var2
30.06.2012    3     4
30.09.2012    5     6
```

Caution : Nothing else could be included in the worksheet as long as the range is not specified!!

- Cross sectional data :

```
types Var1  Var2
var      1     4
std      1     2
```

Caution : Nothing else could be included in the worksheet as long as the range is not specified!!

Output:

- data : An object of class nb_data, nb_ts or nb_cs, where each sheet is added as a separate dataset (page). The sheetnames will appear in the dataNames property of the output.

See also:

[nb_data](#), [nb_ts](#), [nb_cs](#)

Written by Kenneth S. Paulsen

◆ **nb_readExcelMoreSheets**

```
data = nb_readExcelMoreSheets(filename)
```

Description:

Read all the sheets of an excel file given by the input filename and return it as an object of class nb_DataCollection.

Input:

- filename : A string with the filename of the excel spreadsheet.

Output:

- data : An nb_DataCollection object. (Much like an spreadsheet in excel, with each worksheets given as an nb_ts object or an nb_cs objects.)

Format of the worksheets of the excel spreadsheet:

- Timeseries :

dates	Var1	Var2
30.06.2012	3	4
30.09.2012	5	6

Caution : Nothing else could be included in the page!!

- Cross sectional data :

types	Var1	Var2
var	1	4
std	1	2

Caution : Nothing else could be included in the page!!

Output:

- data : An object of class nb_DataCollection

See also:

[nb_DataCollection](#), [nb_ts](#), [nb_cs](#), [nb_readExcel](#)

Written by Kenneth S. Paulsen

◆ **nb_readMat**

data = nb_readMat(filename,sorted)

Description:

This is a function for reading a mat file, and checks if the stored information in that file can be converted to either a nb_data, nb_ts, nb_cs or nb_modelDataSource object

Input:

- filename : A string with the filename of the .mat file.

Formats:

See `nb_data.toStructure`, `nb_ts.toStructure`, `nb_cs.toStructure`,
`nb_data.struct`, `nb_ts.struct` or `nb_cs.struct`

Output:

- data : An object of class nb_data, nb_ts, nb_cs or nb_modelDataSource.

See also:

[nb_ts](#), [nb_cs](#), [nb_data](#), [nb_modelDataSource](#)

Written by Kenneth S. Paulsen

◆ **nb_struct2nb.DataSource**

obj = nb_struct2nb.DataSource(s)

Description:

Convert a struct with nb.DataSource objects as fields to an object of a subclass of the nb.DataSource class.

Input:

- s : A struct with nb.DataSource as fields.

Output:

- obj : A nb.DataSource object

Written by Kenneth Sætherhagen Paulsen

◆ **nb_validpath**

[found,filePath,ext] = nb_validpath(filename,mode)

Description:

Identify if the excel file with name filename is a valid excel file.

Input:

- filename : Must be a string containing a partial path ending in a file or directory name. May contain \ or / or \\. Extension may or may not be included. Tested extensions are '.xls','.xlsx' or '.xlsm'.

- mode : Set to 'write' to prevent it looking for files on the matlab path to locate the path if no path is given. Default is ''.

Output:

```
- found      : true if found, otherwise false.  
- filePath  : A string with the full path of the file.  
- ext       : The file extension. Either '.xls','.xlsx' or '.xlsm'.
```

See also:

[nb_xlsread](#), [nb_xlswrite](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_xlsGetSheets**

```
[sheets,Excel] = nb_xlsGetSheets(file)
```

Description:

Get sheets of an excel file.

Input:

```
- file : Either a filename as a string or a  
        actxserver('Excel.Application') object, see nb_xlsread.
```

Output:

```
- sheets : A cellstr with the sheets of the selected excel file.  
- Excel  : A actxserver('Excel.Application') object.
```

See also:

[nb_xlsread](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_xlsNum2Column**

◆ **nb_xlsread**

```
c = nb_xlsread(filename)  
c = nb_xlsread(filename,sheet)  
c = nb_xlsread(filename,sheet,range)  
c = nb_xlsread(filename,sheet,range,transpose)  
c = nb_xlsread(filename,sheet,range,transpose,mode)  
[c,sheets,Excel] = nb_xlsread(filename,sheet,range,transpose,mode,Excel)
```

Description:

Read an excel file and return an cell matrix representing the selected worksheet.

Caution : xls files

Input:

- filename : The name of the excel file. If extension not provided, .xlsx is assumed.

- sheet : Name of the read sheet. If empty (default) the first sheet of the file is read.

- range : The range to read. Must be given as 'A1:B2', or a 1 x M cell of ranges. If it is given as a cell of ranges the output will also be a cell with the same size as range.

Caution: range must be empty for mode 'nb'.

- mode : > 'default' : Will read the excel sheet as is.

> 'nb' : If the range input is a cell with size 1 x 3 with the range of the dates/types/obs, variables and data.

E.g. {'A2:A5','B1:C1','B2:C5'}

The output will be a cell matrix instead of nested cell. Of course the dimensions of the range must match!

- Excel : Handle to the Excel application. Same as the Excel ouput. It is assumed that a excel file has been opened with this application!

Output:

- c : Either a cell matrix or a nested cell row. See range input.

- sheets : The sheets of the excel spreadsheet, as a cellstr.

- Excel : Handle to the Excel application used for reading the excel spreadsheet. The Excel application has the selected file open for reading!

Examples:

```
c = nb_xlsread('data.xlsx')
```

See also:

[xlsread](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_xlswrite**

```
nb_xlswrite(filename,c,sheet)
```

Description:

Write data to excel spreadsheet.

Input:

- filename : Name of excel file to write to, as a string.
- c : Cell matrix to write to excel
- sheet : The name of the sheet to save the data to, as a string.
- del : Give true to delete default sheet names ('Sheet1','Sheet2','Sheet3'). Default is false.
Caution: This only works for files that are created with this function (for some reason).
- Excel : An Excel application object (actxserver('Excel.Application')). May be smart to only initialize once if writing data to an excel file in a loop!
Caution: It is assumed that the excel file has already been opened by the application for writing!

Examples:

```
nb_xlswrite('test.xlsx',num2cell(rand(3,3)))
```

See also:

[xlswrite](#)

Written by Kenneth Sæterhagen Paulsen

25.3 Overhead graphics classes and functions

- [nb_combineGraphStructDensityGraphs](#)
- [nb_getFigureNumbering](#)
- [nb_graph_bd](#)
- [nb_graph_data](#)
- [nb_graph_package](#)
- [nb_graph_ts](#)
- [nb_localVariables](#)
- [nb_saveas](#)
- [nb_table_cs](#)
- [nb_table_ts](#)
- [nb_getDataNumbering](#)
- [nb_graph_adv](#)
- [nb_graph_cs](#)
- [nb_graph_init](#)
- [nb_graph_subplot](#)
- [nb_localFunction](#)
- [nb_plots](#)
- [nb_table_cell](#)
- [nb_table_data](#)

■ nb_graph_adv

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_adv(plotter,varargin)
```

Superclasses:

handle

Description:

This is a class for making graphs with figure titles and footers.
(MPR styled graphs)

Objects of this class can be given to an object of class
nb_graph_package to create a graph package.

Constructor:

```
obj = nb_graph_adv(plotter,varargin)
```

Input:

- plotter : An object of class nb_graph.
- varargin : 'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_graph_adv

Examples:

```
data      = nb_ts('test');
plotter  = nb_graph_ts(data);
advPlotter = nb_graph_adv(plotter,'figureTitleNor',...
                         'A title','footerNor','A footer');
```

or

```
advPlotter = nb_graph_adv(plotter,'figureTitleNor',...
                           char('A title','A new line'),...
                           'footerNor',...
                           char('A footer','A new line'));
```

or

```
advPlotter = nb_graph_adv(plotter,'figureTitleNor',...
                           {'A title','A new line'},...
                           'footerNor',...
                           {'A footer','A new line'});
```

See also:

[nb_figureTitle](#), [nb_footer](#), [nb_graph_ts](#), [nb_graph_cs](#), [nb_ts](#), [nb_cs](#)

nb_graph_package

Written by Kenneth Sæterhagen Paulsen

Properties:

- a4Portrait
- chapter
- counter
- defaultFigureNumbering
- excelFooterEng
- excelFooterNor
- figureNameEng
- figureNameNor
- figureTitleAlignment
- figureTitleFontSize
- figureTitleInterpreter
- figureTitlePlacement
- figureTitleWrapping
- flip
- fontName
- fontUnits
- footerAlignment
- footerEng
- footerFontSize
- footerFontWeight
- footerInterpreter
- footerNor
- footerPlacement
- footerWrapping
- forecastDate
- forecastTypes
- jump
- legendsEng
- letter
- letterRestart
- localVariables
- number
- plotter
- remove
- roundoff
- saveName
- tooltipEng
- tooltipNor
- tooltipWrapping
- userData

- a4Portrait ↑

Save PDF to A4 portrait format. 0 or 1

- chapter ↑

Overrun the chapter property of the nb_graph_package it is included in. Default is [], i.e. use the chapter of the nb_graph_package. Must be set to a number.

- counter ↑

Sets the figure counter to this number when included in a nb_graph_package object. Nice property to have if some graphs of the package is created with other tools. Must be a scalar.

- **defaultFigureNumbering** ↑

Set to true to add Figur 1 to the first line of figureTitleNor and Graph 1 to figureTitleEng.

- **excelFooterEng** ↑

Deprecated

- **excelFooterNor** ↑

Deprecated

- **figureNameEng** ↑

Sets the english name of the figure. This name will be used on the englsg index page of the excel spreadsheet.

- **figureNameNor** ↑

Sets the norwegian name of the figure. This name will be used on the norwegian index page of the excel spreadsheet.

- **figureTitleAlignment** ↑

Sets the alignment of the figure title. As a string. Either 'left', 'center' or 'right'. 'left' is default.

- **figureTitleEng** ↑

Sets the english figure title of the graph. Either a string ('A figure title'), char (char('A figure title', 'A new line')) or a cellstr ({'A figure title', 'A new line'})

6 lines of the figure title is supported.

- **figureTitleFontSize** ↑

Sets the figure title font size. 0.0511 is default.

- **figureTitleFontWeight** ↑

Sets the figure title font weight. Must be a string. Either 'bold', 'light', 'normal' or 'demi'. 'normal' is default

- **figureTitleInterpreter** ↑

Sets the interpreter of the figure title text. Must be a string. Either 'latex' (mathematical expressions), 'tex' (Normal latex interpreter) or 'none' (No interpreter). Default is 'tex'.

- **figureTitleNor** ↑

Sets the norwegian figure title of the graph. Either a string ('A figure title'), char (char('A figure title', 'A new line')) or a cellstr ({'A figure title', 'A new line'})

6 lines of the figure title is supported.

- **figureTitlePlacement** ↑

Sets where to place the figure title in the x-axis direction, either 'center', 'leftaxes', 'right' or the default value 'left'.

- **figureTitleWrapping** ↑

Wrap text of figure title automatic. Default is false, i.e. no wrapping.

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

- **fontName** ↑

Sets the font name used for the text of graph. As a string. Default is 'arial'.

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : When the legType is set to 'MATLAB' this property will not set the font units of the legend.

Caution : Don't use this property in combination with one of the graph styles defined by this class. I.e. 'mpr', 'mpr_white', 'presentation' and

'presentation_white' They will set this property to 'normalized'.

Caution : 'normalized' font units must be between 0 and 1.
The normalized units are not the same as the MATLAB
normalized units. This normalized units are not
only normalized to the axes size, but also across
resolution of the screen. (But only vertically)

- **footerAlignment** ↑

Sets the alignment of the footer. As a string. Either
'left', 'center' or 'right'. 'left' is default.

- **footerEng** ↑

Sets the english footer of the graph. Either a string
('A footer'), char (char('A footer','A new line')) or
a cellstr ({'A footer','A new line'})

6 lines of the footer is supported.

- **footerFontSize** ↑

Sets the footer font size. 0.0397 is default.

- **footerFontWeight** ↑

Sets the footer font weight. Must be a string. Either
'bold', 'light', 'normal' or 'demi'. 'normal' is default.

- **footerInterpreter** ↑

Sets the interpreter of the footer text. Must be a string.
Either 'latex' (mathematical expressions), 'tex' (Normal
latex interpreter) or 'none' (No interpreter). Default is
'tex'.

- **footerNor** ↑

Sets the norwegian footer of the graph. Either a string
('A footer'), char (char('A footer','A new line')) or a
cellstr ({'A footer','A new line'})

6 lines of the footer is supported.

- **footerPlacement** ↑

Sets where to place the footer in the x-axis direction,
either 'center', 'leftaxes', 'right' or the default
value 'left'.

- **footerWrapping** ↑

Wrap text of footer automatic. Default is false, i.e. no wrapping.

- **forecastDate** ↑

Sets from where to color the text of the excel output file as forecast. Either a string with the date ('2012Q1') or a cellstr ({'Var1','2012Q1','Var2','2012Q2'}).

- **forecastTypes** ↑

Sets from where to color the text of the excel output file as forecast. Either a cellstr with the types which is "forecast".

- **jump** ↑

Sets the number of graphs the counter should jump. As a scalar. Default is 1.

- **legendsEng** ↑

Sets the english legends of the produced graph. I.e. overrun the legends property of the given nb_graph_ts or nb_graph_cs object. Must be a cellstr. I.e. {'legend1','legend2'}

Caution : This will set the legAuto property of the nb_graph_ts or nb_graph_cs object to 'off'. I.e. You must also provided the legends for the patches and fake legends by this property. See the documentation of the nb_graph_ts or nb_graph_cs for more on the legAuto property.

- **letter** ↑

Give 1 if the graph should be "numbered" with a letter instead of an number. I.e. 'Figur 1.1a' or 'Chart 1.1a'. Default is 0 (Numbering the graph as 'Figur 1.1' or 'Chart 1.1')

- **letterRestart** ↑

Restart letter counting. {0} (false) | 1 (true).

- **localVariables** ↑

A handle to an nb_struct object storing the local variables

- **number** ↑

Sets the number of the graph. As a scalar. E.g. 'Figure 1' or 'Chart 1' if the number property is set to 1.

- **plotter** ↑

Sets the underlying graph object. Must be of class nb_graph_ts, nb_graph_cs, nb_graph_data, nb_table_ts, nb_table_cs nb_table_data or nb_table_cell.

- **remove** ↑

Sets the variables which can not be published. I.e. will not written to the excel output file. Must be a cellstr. E.g. {'Var1','Var2',...}.

- **roundoff** ↑

When included in a nb_graph_package object this property will overrun the nb_graph_package.roundoff property if not empty. Must be an integer greater or equal to 0, or empty (uses the property of the package). Default is empty.

- **saveName** ↑

Sets the savename of the saved output file. Mainly used by the nb_graph_package class. Must be a string.

- **tooltipEng** ↑

Text to be displayed as a tooltip with technical comments when reading electronic report. If empty, excel footer is used when writing text. English.

- **tooltipNor** ↑

Text to be displayed as a tooltip with technical comments when reading electronic report. If empty, excel footer is used when writing text. Norwegian.

- **tooltipWrapping** ↑

Boolean indicating whether to wrap the tooltip. Default is false.

- **userData** ↑

User data, can be set to anything you want.

Methods:

- | | | | |
|--|--------------------------------|-------------------------------|------------------------------|
| • addGraph | • getData | • graphEng | • graphNor |
| • helpAdvancedMenuCallback | • realTimespan | • removeGraph | • saveFigure |
| • set | • setSpecial | • timespan | • update |

► **addGraph** ↑

```
addGraph(obj,another)
```

Description:

Add another graph to a 1 x 2 panel.

Input:

- obj : A nb_graph_adv object.
- another : A nb_graph object.

Written by Kenneth Sæterhagen Paulsen

► **getData** ↑

```
data = getData(obj)
```

Description:

Get the data of the graph

Input:

- obj : An object of class nb_graph_adv

Output:

- data : As a nb_ts, nb_cs or nb_data object.

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

► **graphEng** ↑

```
graphEng(obj)
```

Description:

Plot the graph (english version)

Input:

- obj : An object of class nb_graph_adv

Output:

The graph plotted on the screen (english version)

Examples:

```
obj.graphEng()
```

Written by Kenneth S. Paulsen

► **graphNor** ↑

```
graphNor(obj)
```

Description:

Plot the graph (norwegian)

Input:

- obj : An object of class nb_graph_adv

Output:

The graph plotted on the screen (norwegian version)

Examples:

```
obj.graphNor()
```

Written by Kenneth S. Paulsen

► **helpAdvancedMenuCallback** ↑

```
nb_graph_adv.helpAdvancedMenuCallback(uiHandle, event)
```

Description:

Part of DAG

Opens a nb_helpWindow with useful comments on the advanced graph menu.

Written by Per Bjarne Bye

► **realTimespan** ↑

```
dateString = realTimespan(obj)
```

Description:

Get the timespan of the data behind a graph which is not nan or infinite, as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_ts

Output :

- dateString : A string with the timespan of the graph represented by this object. The language of this string will depend on the language of the language property of the plotter object.

Written by Kenneth S. Paulsen

► **removeGraph** ↑

```
removeGraph(obj)
```

Description:

Add another graph to a 1 x 2 panel.

Input:

- obj : A nb_graph_adv object.
- another : A nb_graph object.

Written by Kenneth SÃ!terhagen Paulsen

► **saveFigure** ↑

```
saveFigure(obj)
```

Description:

Save the figure to a pdf file

Input:

- obj : An object of class nb_graph_adv

Output:

The plotted graph saved to a pdf file, given by the saveName property of the object.

Examples:

```
obj.saveFigure()
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of the nb_graph_adv object.

See the documentation of the nb_graph_adv class for more on the different properties.

Input:

- obj : An object of class nb_graph_adv
- varargin : 'propertyName',PropertyValue,...

Output:

- obj : The input object with the updated properties.

Examples:

```
set(obj,'propertyName',PropertyValue);
obj.set('propertyName',PropertyValue,...)
obj = obj.set('footerfontsize',14);
```

Written by Kenneth S. Paulsen

► **setSpecial** ↑

```
setSpecial(obj,varargin)
```

Written by Kenneth S. Paulsen

► **timespan** ↑

```
dateString = timespan(obj)
```

Description:

Get the timespan of the graph as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_adv, with the property plotter set to an nb_graph_ts object. Will return an empty string otherwise.

Output :

- dateString : A string with the timespan of the graph represented by this object. The language of this string will depend on the language of the language property of the plotter object.

Examples:

Written by Kenneth S. Paulsen

► **update ↑**

```
obj = update(obj)
```

Description:

Update the data source of the nb_graph_adv object. I.e update the data source of the plotter property.

See the update method of the nb_cs or nb_ts class for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_graph_adv

Output:

- obj : An object of class nb_graph_adv, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ nb_graph_bd

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_bd(varargin)
```

Superclasses:

handle, nb_graph

Description:

This is a class for making business days (sparse time-series) graphics.

Constructor:

```
obj = nb_graph_bd(data)
```

Input:

- data : The input must one of the following:

> An object of class nb_bd. E.g. nb_graph_bd(data)

> A excel spreadsheet which could be read by the nb_bd class. E.g. nb_graph_bd('excelName1')

It is also possible to initialize an empty nb_graph_bd object. (Do not provide any inputs to the constructor)

Output:

- obj : An object of class nb_graph_bd

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_data](#), [nb_graph_cs](#), [nb_graph_adv](#), [nb_graph_subplot](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- DB
- a4Portrait
- addSpace
- annotation
- areaAccumulate
- axesFast
- axesFontSizeX
- axesLineWidth
- axesScaleFactor
- barAlpha1
- barBlend
- barShadingColor
- barShadingDirection
- baseLine
- baseLineStyle
- baseValue
- candleIndicatorLineStyle
- candleVariables
- code
- colorOrder
- colors
- dashedLine
- dateFormat
- displayValue
- excelFooterEng
- excelTitleEng
- factor
- GraphStruct
- addAdvanced
- alignAxes
- areaAbrupt
- areaAlpha
- axesFontSize
- axesFontWeight
- axesPrecision
- axesScaleLineWidth
- barAlpha2
- barLineWidth
- barShadingDate
- barWidth
- baseLineColor
- baseLineWidth
- candleIndicatorColor
- candleMarker
- candleWidth
- colormap
- colorOrderRight
- crop
- dataType
- dateInterpreter
- endGraph
- excelFooterNor
- excelTitleNor
- fakeLegend

- fanAlpha
- fanColor
- fanDatasets
- fanFile
- fanGradedLineWidth
- fanGradedStyle
- fanLegend
- fanLegendFontSize
- fanLegendLocation
- fanLegendText
- fanMethod
- fanPercentiles
- fanVariable
- fanVariables
- figureColor
- figureName
- figurePosition
- figureTitle
- fileFormat
- findAxisLimitMethod
- flip
- fontName
- fontScale
- fontUnits
- graphStyle
- grid
- gridLineStyle
- highlight
- highlightColor
- horizontalLine
- horizontalLineColor
- horizontalLineWidth
- language
- legAuto
- legBox
- legColor
- legColumnWidth
- legColumns
- legFontColor
- legFontSize
- legInterpreter
- legLocation
- legPosition
- legReorder
- legSpace
- legendText
- legends
- lineStyles
- lineWidth
- lineWidths
- localVariables
- lookUpMatrix
- mYLim

- markerSize
- missingValues
- noLegend
- noTickMarkLabelsLeft
- noTitle
- numberOfGraphs
- parameters
- patchAlpha
- plotAspectRatio
- plotTypes
- saveName
- scatterDatesRight
- scatterVariables
- shading
- startGraph
- subPlotSpecial
- sumTo
- title
- titleFontSize
- titleInterpreter
- userData
- variablesToPlotRight
- verticalLineColor
- verticalLineStyle
- xLabel
- xLabelFontSize
- xLabelInterpreter
- markers
- noLabel
- noTickMarkLabels
- noTickMarkLabelsRight
- normalized
- page
- patch
- pdfBook
- plotType
- position
- scatterDates
- scatterLineStyle
- scatterVariablesRight
- spacing
- subPlotSize
- subPlotsOptions
- tag
- titleAlignment
- titleFontWeight
- titlePlacement
- variablesToPlot
- verticalLine
- verticalLineLimit
- verticalLineWidth
- xLabelAlignment
- xLabelFontWeight
- xLabelPlacement

- `xLim`
- `xTickLabelAlignment`
- `xTickLabels`
- `xTickRotation`
- `yDirRight`
- `yLabelFontSize`
- `yLabelInterpreter`
- `yLim`
- `yOffset`
- `yScaleRight`
- `ySpacing`
- `ySpacingRight`
- `xScale`
- `xTickLabelLocation`
- `xTickLocation`
- `yDir`
- `yLabel`
- `yLabelFontWeight`
- `yLabelRight`
- `yLimRight`
- `yScale`
- `ySpacing`

- **`DB`** 

All the data given to the `nb_graph_ts` object collected in a `nb_bd` object. Should not be set using `obj.DB = data!` Instead see the `nb_graph_bd.resetDataSource` method.

- **`GraphStruct`** 

Sets the information on which variables and how to plot them when using the `graphInfoStruct(...)` method of this class.

Must be an .m-file with the code on how the structure of graphing information should be initialized or as a structure with the graphing information.

Only the `graphInfoStruct(...)` method uses this property. See the documentation of the NB toolbox for more on this input.

Inherited from superclass `NB_GRAPH`

- **`a4Portrait`** 

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass `NB_GRAPH`

- **`addAdvanced`** 

Set to false to prevent adding advanced components.

Inherited from superclass `NB_GRAPH`

- **`addSpace`** 

Sets the space before and/or after in the x direction of the graph [`addedSpaceBefore` `addedSpaceAfter`]. E.g. add one period before and after [1,1].

Inherited from superclass `NB_GRAPH`

- **alignAxes** ↑

Align axes at a given base value. Must be set to a scalar double.
E.g. 0.

Inherited from superclass NB_GRAPH

- **annotation** ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the nb_annotation class for more. If you give more objects they must be collected in a cell array.

Examples can be found here:

\NBTOOLBOX\Examples\Graphics\nb_annotationExamples.m

Inherited from superclass NB_GRAPH

- **areaAbrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan. Default is false.

Inherited from superclass NB_GRAPH

- **areaAccumulate** ↑

Set if the areas should be accumulated or not. Default is true.

Inherited from superclass NB_GRAPH

- **areaAlpha** ↑

Sets the transparency of the area chart. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **axesFast** ↑

Set the if you want graphs to be produced fast with a loss of functionality and look or not. When empty default for the graph method is false, while for the graphSubPlots and graphInfoStruct methods are true.

Inherited from superclass NB_GRAPH

- **axesFontSize** ↑

Sets the font size of the axes tick mark labels, default is 12. Must be a scalar.

Inherited from superclass NB_GRAPH

- **axesFontSizeX** ↑

Sets the font size of the x-axis tick mark labels, default is []. I.e. use the axesFontSize property instead. Must be a scalar.

This property will not be scaled when fontUnits are changed, as is the case for axesFontSize.

Caution: Will not set the x-axis font size when plotType is set to scatter.

Inherited from superclass NB_GRAPH

- **axesFontWeight** ↑

Sets the font weight of the axes tick mark labels. Must be a string. Default is 'normal'. Either 'normal', 'bold', 'light' or 'demi'

Inherited from superclass NB_GRAPH

- **axesLineWidth** ↑

Sets the the line width of the axes, default is 0.5. Must be a set to a scalar double greater than 0.

Inherited from superclass NB_GRAPH

- **axesPrecision** ↑

Sets the precision/format of the rounding of number on the axes.

See the precision input to the nb_num2str function. Default is [], i.e. to call num2str without additional inputs.

Inherited from superclass NB_GRAPH

- **axesScaleFactor** ↑

Set the scaling factor used when axesScaleLineWidth is set to true.

Inherited from superclass NB_GRAPH

- **axesScaleLineWidth** ↑

Scale line width to axes height. true or false (default). If axesScaleFactor is set to a scalar number this will be the scaling factor used instead of the automatic scaling factor.

Inherited from superclass NB_GRAPH

- **barAlpha1** ↑

Set the alpha blending parameter 1, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barAlpha2** ↑

Set the alpha blending parameter 2, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barBlend** ↑

Set to true to do alpha blending instead of shading if shading options is used. Default is false.

Inherited from superclass NB_GRAPH

- **barLineWidth** ↑

Set the bar line width without affecting any other line widths. Default is empty, i.e. use the lineWidth property.

Inherited from superclass NB_GRAPH

- **barShadingColor** ↑

The color shaded bars are interpolated with. Default is 'white'. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **barShadingDate** ↑

Sets the date from which the bar plot should be shaded. Either a string or an object which is of a subclass of the nb_date class.

- **barShadingDirection** ↑

Sets the angle the shading should be vertical of. Either {'north'} | 'south' | 'west' | 'east'. As a string. Default is 'north'.

Inherited from superclass NB_GRAPH

- **barWidth** ↑

Sets the width of the bar plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **baseLine** ↑

If 0 is given the base line will not be plotted, otherwise it will be plotted.

Inherited from superclass NB_GRAPH

- **baseLineColor** ↑

Sets the color of the base line. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **baseLineStyle** ↑

Sets the line style of the base line. As a string. Either {'-' } | '--' | '---' | ':' | '-.' | 'none'.

Inherited from superclass NB_GRAPH

- **baseLineWidth** ↑

Sets the width of the base line. Must be a scalar.

Inherited from superclass NB_GRAPH

- **baseValue** ↑

Sets the base value used for bar and area plot. (Sest also the y-axis base value for the base line)

Inherited from superclass NB_GRAPH

- **candleIndicatorColor** ↑

Sets the color of the indicator of the candle. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **candleIndicatorLineStyle** ↑

Sets the line style of the indicator of the candle. Must be a string. Default is '-'.

Inherited from superclass NB_GRAPH

- **candleMarker** ↑

Sets the marker of the indicator of the candle. Must be a string. Default is 'none'.

Inherited from superclass NB_GRAPH

- **candleVariables** ↑

Sets which variables are going to be plotted as open, close, high, low and indicator. Must be a cellstr. E.g.

{'open','var1','close','var2',...}

If a type (e.g. 'low') is not given it will not be plotted.

Meaning of each type:

- > 'close' : The lowest values of the plotted patches of the candle.
- > 'high' : The highest values of the plotted candles.
- > 'indicator' : The value for where to plot a horizontal line (across the patch). E.g. the mean value.
- > 'open' : The highest values of the plotted patches of the candle.
- > 'low' : The lowest values of the plotted candles.

Inherited from superclass NB_GRAPH

- **candleWidth** ↑

Sets the width of the candle plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **code** ↑

If you set this to a file name, this file will be evaluated after all the plotting is done. Must be a string.

This could be nice for adding special annotation, lines and so on. In this file you can use any MATLAB function, but some strange error message can occur if you use local variables which is already defined (So use long and descriptive variable names in the provided code to prevent this).

This property is only an option for the method graph.

Nice to know:

> The current nb_figure object can be reached with
get(obj,'figureHandle')

> The current nb_axes object can be reached with
get(obj,'axesHandle')

Caution : Be aware that the name of the file must be unique and cannot be a MATLAB function or a local variable.

Caution : The file must be a MATLAB .m file

Inherited from superclass NB_GRAPH

- **colorMap** ↑

Sets the colormap used when plotType is set to 'image' and fanMethod is set to 'graded'. Must be given as n x 3 double or the path to a .mat file that contain the colormap. Default is given by nb_axes.defaultColorMap.

For an example of a supported MAT file see:

- ...\\Examples\\Graphics\\colorMapNB.mat

Inherited from superclass NB_GRAPH

- **colorOrder** ↑

Sets the color order of the plotted data (Left axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the left axes.

Inherited from superclass NB_GRAPH

- **colorOrderRight** ↑

Sets the color order of the plotted data (Right axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the right axes.

Inherited from superclass NB_GRAPH

- **colors** ↑

Sets the colors of the given variable(s). Must be a cell array on the form: {'var1', 'black', 'var2', [0.2 0.2 0.2],...}. Here we set the colors of the variables 'var1' and 'var2' to 'black' and [0.2 0.2 0.2] (RGB color) respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Caution: For the graphInfoStruct() method you need to select the variables from obj.DB.dataNames instead!

Inherited from superclass NB_GRAPH

- **crop** ↑

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_GRAPH

- **dashedLine** ↑

Sets the date at which all the plotted lines will be dashed. Must be given as a string or an object which is of a subclass of the nb_date class.

- **dataType** ↑

The data type to plot, either 'stripped' (default) or 'full'. If 'stripped' only periods where any of the variables have data is plotted, otherwise all periods of the timespan of the data is plotted.

- **dateFormat** ↑

The date format used for the dates on the x-axis labels. Default is 'nbNorwegian' for norwegian and 'nbEnglish' for english (i.e. when set to '').

- **dateInterpreter** ↑

Sets how to interpreter the dates. I.e. where to place the datapoints in relation to the tick mark labels. Must be a string. {'start'} | 'end' | 'middle'.

- **displayValue** ↑

Set to false to not display observation values when holding the mouse over the graph. Default is true.

Inherited from superclass NB_GRAPH

- **endGraph** [↑](#)

Sets the end date of the graph. Either as string or an object which is of a subclass of the nb_date class. If the given date is after the end date of the data, the data will be appended with nan values. I.e. the graph will be blank for these dates.

- **excelFooterEng** [↑](#)

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. English version.

Inherited from superclass NB_GRAPH

- **excelFooterNor** [↑](#)

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. Norwegian version.

Inherited from superclass NB_GRAPH

- **excelTitleEng** [↑](#)

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). English version.

Inherited from superclass NB_GRAPH

- **excelTitleNor** [↑](#)

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). Norwegian version.

Inherited from superclass NB_GRAPH

- **factor** [↑](#)

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_GRAPH

- **fakeLegend** [↑](#)

Set this property if you want to add a legend of none plotted variable. You can set this property. Must be a cell on the form: {'legendName', settings,...}, where the legendName will be text of the fake legend. And the settings input must also be a cell array with the optional inputs {...,'propertyName', propertyName,...}. These settings will set the appearance of the fake legend. The following propertyNames are supported:

```
> 'type'      : Either 'line' or 'patch'.

> 'color'     : Sets the color of the fake legend. Default is
                 'black'. Must either be a 1 x 3 double with
                 the RGB colors or a string with the color
                 name. If shaded patch is wanted it must be
                 2 x 3 double with the RGB colors or a 1 x 2
                 cellstr with the color names to use.

> 'direction' : Sets the direction of the shading. Either
                 {'north'} | 'south' | 'west' | 'east'. As a
                 string. Default is 'north'. Only an option
                 when 'type' is set to 'patch'.

> 'edgeColor'  : Sets the color of the edge of the patch.
                 Default is the same as the 'color' option.
                 Only an option when 'type' is set to 'patch'.

> 'lineStyle'   : Sets the line style of the fake legend. As a
                 string. Either {'-' } | '--' | '---' | ':' |
                 '-.' | 'none'. Default is a normal line.

                 Will set the edge line style if the 'type'
                 is set to 'patch'.

> 'lineWidth'   : Sets the line width of the fake legend.
                 Default is 2.5. Must be a scalar.

> 'marker'     : Sets the marker of the fake legend. Default is
                 'none'. Only an option when 'type' is set to
                 'line'. Lookup LineSpec in the MATLAB help
                 meny for more on the supported marker types.
```

Caution : Even if you don't want to set any of the optional input. The settings input must be given. I.e an empty cell, i.e. {}.

Inherited from superclass NB_GRAPH

- **fanAlpha** ↑

Sets the transparency of the fan chart. May be needed if you add fan charts to more than one variable at the time. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

- **fanColor** ↑

Sets the colors of the fan charts. Must be a matrix of size; number of fan layer x 3, with the RGB color specifications.

or

A string with the basis color for the fan colors. Either 'red', 'blue', 'green', 'yellow', 'magenta', 'cyan' or 'white'.

Be aware that the classification of the color of the given "basecolor" is based on the property fanPercentiles, so if you have more or less than 4 layers you must change this property to have the same size as the number of layers. (This is the case even if you don't use it to calculate the percentiles of a given datasets. Given by the property fanDatasets)

Default is the nb colors.

Caution: If fanMethod is set to 'graded' the color used are those assign to the colorMap property.

- **fanDatasets** ↑

Sets the data of the added fan layers.

If you have the data of the different layer in separate datasets. Then the input to this property must be a 2 * number of layers cell matrix. Can be excel spreadsheets, .mat files or dyn_ts objects.

or

You can also give the data with all the simulated data or fan layers in an dyn_ts object or an nb_ts object. (Each page one simulation or each page one layer). (FASTEST with an nb_ts object as input.)

Remember that the percentiles for the fan charts is set by the property fanPercentiles, default is [.3, .5, .7, .9].

The fan layers are added differently given the method you use.

> graph(...): Adds the fan given in this property to the last variable given in the property variablesToPlot. (If not the fanVariable property is set.) If you graph more datasets (in separate figures), the fan layers will be the same for all of them. Set the fanVariables property if that is not wanted.

> graphSubPlots(...): Adds the fan given in this property to the corresponding variable in the last given datasets (last page of the nb_ts object given by the property DB) of each each subplot.

-graphInfoStruct(...): Adds the fan given by this property to each variable of each subplot. Added for the last given dataset. (Last page of the nb_ts object given by the

property DB.)

- **fanFile** ↑

The file to get the default fan layers from. As a string.

- **fanGradedLineWidth** ↑

Line width used but the graded fan chart is set to

- **fanGradedStyle** ↑

Line width used but the graded fan chart is set to

- **fanLegend** ↑

Set this property if you want to add a MPR styled legend of the fan layers. Set it to 1 if wanted. Default is not (0).

- **fanLegendFontSize** ↑

Sets the font size of the fan legend. Default is 12. Must be a scalar.

- **fanLegendLocation** ↑

Sets the location of the fan legend. Locations are:

```
> 'Best', (same as 'North'.)
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'outside' : Places the legend outside the axes, one the right side. (If you create subplots, this is the only option which will look good.)
```

You can also give a 1 x 2 double vector with the location of where to place the legend. First column is the x-axis location and the second column is the y-axis location. Both must be between 0 and 1.

- **fanLegendText** ↑

Set this property if you want to give the text over the colored boxes of the MPR looking fan legend, you must give them as a cell. (Must have same size as number of layers.) Default is the percentiles with the % sign added. E.g. {'30%','50%','70%','90%'}.

- **fanMethod** ↑

{'percentiles'} | 'hdi' | 'graded'; Which type of fan chart is going to be made. 'percentiles' are produced by using percentiles, 'hdi' use highest density intervals, while 'graded' creates a graded fan chart of the colors specified with the colorMap property.

- **fanPercentiles** ↑

Sets the percentiles you want to calculate (if the fanDatasets property is representing simulation) or specify the percentiles you have given (if the fanDatasets property is representing already calculated percentiles). Must be given as a double vector. Default is [0.3 0.5 0.7 0.9].

This property is also the default base for the text of the fan legend. (Where the percentiles are given in percent.)

- **fanVariable** ↑

Sets the variable you want to add the fan layers to. Must be given as a string with the variable name. Default is to use the last variable in the variablesToPlot property.

If this input is a cellstr with more than one variable, the 'fanPercentiles' input must be a scalar double. Fan chart will then be added to more than one variable.

- **fanVariables** ↑

If you have the fan layers included in the same dataset as your other plotted variables you can use this property. The input must then be given as 1 x 2 * number of layers cell array with the fan layers names (as strings). I.e. both the upper and lower layers must be given as separate variables.
(Need not be ordered though.)

E.g. {'LowPerc 90%', 'LowPerc 70%', 'UppPerc 90%', 'UppPerc 70%'}

Caution: Must fit the number of percentiles given by the property fanPercentiles (Remember to include both upper and lower bounds!).

Only an input to the method graph(...): Adds the given fan layers to the last variable given by the property variablesToPlot or the variable given by fanVariable property.

- **figureColor** ↑

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_GRAPH

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_GRAPH

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_GRAPH

- **figureTitle** ↑

Sets if figure titles is wanted on the figures when using the method graphInfoStruct. The fieldnames of the GraphStruct property will be used as figure title. Default is 0 (not include).

If set to a one line char this property can be used for the plotting method graphSubPlots.

Inherited from superclass NB_GRAPH

- **fileFormat** ↑

Sets the save file format. Either 'eps', 'jpg', 'pdf' or 'png'. 'pdf' is default.

Inherited from superclass NB_GRAPH

- **findAxisLimitMethod** ↑

Sets the method used for finding the axis limits, and y-ticks.

- > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
- > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
- > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
- > 4 : Uses the MATLAB algorithm for finding the axis limits.

Inherited from superclass NB_GRAPH

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_GRAPH

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_GRAPH

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_GRAPH

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either
{'points'} | 'normalized' | 'inches' | 'centimeters'.
Default is 'points'.

Caution : If you set the fontUnits property in a method call
to the set method, all the font properties set
before the fontUnits property will be set in the
old fontUnits, while all the font properties set
after will be in the new fontUnits.

Caution : Don't use this property in combination with one of
the graph styles defined by this class. I.e.
'mpr', 'mpr_white', 'presentation' and
'presentation_white' They will set this property
to 'normalized'.

Caution : 'normalized' font units must be between 0 and 1.
The normalized units are not the same as the MATLAB
normalized units. This normalized units are not
only normalized to the axes size, but also across
resolution of the screen. (But only vertically)

Inherited from superclass NB_GRAPH

- **graphStyle** ↑

Sets the graphing style of the plots. Give 'mpr' if you want
the MPR looking style, or 'presentation' for
graphs for the presentation (see nb_Presentation). If you
don't want the grey shaded background you can use 'mpr_white'
and 'presentation_white' instead.

You can also add your own graph style setting through a .m
file. I.e. give the filename (without extension) as a string.

E.g. In the file you can type in something like:

```
set(obj,'titleFontSize',12,'legInterpreter','tex');
```

Caution: If you give this as the first input to the set method
it is possible to overrun some of the default
settings if that is wanted. E.g. font size of the
text, the shading of the plot and so on.

Inherited from superclass NB_GRAPH

- **grid** ↑

Set it to 'on' if grid lines are wanted otherwise set it to
'off'. Default is 'off'.

Inherited from superclass NB_GRAPH

- **gridLineStyle** ↑

Sets the line style of the grid lines. Either '-' , '--' , ':'
or '-.'.

Inherited from superclass NB_GRAPH

- **highlight** ↑

If highlighted area between two dates of the graph is wanted set this property.

One highlighted area:

A cell array on the form {'date1', 'date2'}

More highlighted areas:

A nested cell array on the form

{{'date1(1)', 'date2(1)'}, {'date1(2)', 'date2(2)'}, ...}

If plotType is set to 'scatter' scalars can be used instead.

- **highlightColor** ↑

Sets the colors of the background patches. Must be one of the following:

- > A string with the color name to use when plotting (all) the patch(es)
- > A cell array of strings with the color names
- > A cell with 1 x 3 doubles with the RGB color specification.

When given as a cell this property must match the highlight property. I.e. the number of highlighted areas added.

The default color of the highlighted areas is 'light blue'.

- **horizontalLine** ↑

If you want to include some extra horizontal lines, beside the base line, you can set this property. Must be set to a scalar or a double vector with the y-axis value(s) on where to place the horizontal line(s).

Inherited from superclass NB_GRAPH

- **horizontalLineColor** ↑

Sets the color(s) of the given horizontal line(s).

Must either be an M x 3 double with the RGB color(s) or a 1 x M cell array of strings with the color name(s). Where M must be less than the number of plotted horizontal lines.

If less or no color(s) is/are set by this property the default colors for the rest or all the horizontal line(s) is/are [0 0 0], i.e. MATLAB black.

Inherited from superclass NB_GRAPH

- **horizontalLineStyle** ↑

Sets the style(s) of the given horizontal line(s).

Must either be a string or a 1 x M cell array of strings with the style(s). Where M must be less than the number of plotted

horizontal lines.

If less or no style(s) is/are set by this property the default lines for the rest or all the horizontal line(s) is/are '-'.

Inherited from superclass NB_GRAPH

- **horizontalLineWidth** ↑

Sets the line width of the horizontal line(s). Must be a scalar. Default is 1.

Inherited from superclass NB_GRAPH

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_GRAPH

- **legAuto** ↑

'on' | {'off'}. As a string.

> 'on' : The strings given through the fakeLegend and patch properties are automatically added to the legend.

> 'off' : The strings given through the fakeLegend and patch properties are not automatically added to the legend. Which means that you must provide all the legend information through the legends property. The patch descriptions must be given first and the fake legend description must be given last in the legends property. Default.

Inherited from superclass NB_GRAPH

- **legBox** ↑

Set if a box should be drawn around the legend. Either {'on'} | 'off'.

Caution : When removing the box you make it impossible to move it around afterwards.

Inherited from superclass NB_GRAPH

- **legColor** ↑

Sets the color of the background of the box of the legend. Either as 1x3 double with the RGB or as a string). Default is 'none', i.e. transparent.

Inherited from superclass NB_GRAPH

- **legColumnWidth** ↑

Sets the width of the columns of the legend. Default is to let MATLAB find the width itself. If given it must be a scalar with the width of all columns of the legend or a $1 \times \text{legColumns}$ double with the width of each individual column.

Should be between 0 and 1. Use the try and fail method to find out what fits.

Inherited from superclass NB_GRAPH

- **legColumns** [↑](#)

Sets the number of columns of the legends. Must be a scalar.

Inherited from superclass NB_GRAPH

- **legFontColor** [↑](#)

Sets the font color(s) of the legend text. Must either be a 1×3 double or a string with the color name of all legend text objects, or a $M \times 3$ double or a cellstr array with size $1 \times M$ with color names to use of each legend text object.

Inherited from superclass NB_GRAPH

- **legFontSize** [↑](#)

Sets the font size of the legend. Must be a scalar default is 10.

Inherited from superclass NB_GRAPH

- **legInterpreter** [↑](#)

Sets the interpreter of the legend text. Must be a string.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **legLocation** [↑](#)

Sets the location of the legend, must be string. The location can be:

> 'Best' : Same as 'NorthWest'.
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'below' : Below the plot (when many subplots, it will place the legend below all of them)
> 'middle' : Could only be use for 2×2 subplot, and then it places the legend between the upper and lower subplots.

Location that only works for the graph() method:

```

> 'outsideright'      : Legend is placed outside the axes on the
                         right side, and the axes is resized, so
                         the axes and legend does not take up more
                         space than the axes did in the first place.
                         The legend is vertically centered.
> 'outsiderighttop'   : Same as 'outsideright', but placed
                         vertically top.

```

Inherited from superclass NB_GRAPH

- **legPosition** ↑

Sets the position of the legend. Must be a 1x2 double. Where the first element is the x-axis location and the second is the y-axis location. [xPosition yPosition]

Caution : Both elements must be between 0 and 1. And where we have that [0, 0] will be the bottom left location of the axes and [1, 1] is the top right position of the axes. (This position will be the top left position of the legend itself.)

Caution : This option overruns the legLocation property.

Inherited from superclass NB_GRAPH

- **legReorder** ↑

Set this property to reorder the legend. Either a string with {'default'} | 'inverse' or a double with how to reorder the legend. E.g. if you have 3 legends to plot the default index will be [1,2,3], the inverse ('inv') will be [3,2,1]. If you want to decide that the 3. legend should be first, then the 1. and 2., you can type in [3,1,2].

Caution : If a double is given it must have as many elements as there is plotted legends. I.e. if you give a empty string to one of the legends it will be excluded. Say you provide the property legends as {'s','t','f'} then the double must have size 1x3.

Inherited from superclass NB_GRAPH

- **legSpace** ↑

The vertical space between the texts of the legend. Must be a scalar. Default is 0.

Inherited from superclass NB_GRAPH

- **legendText** ↑

Sets the legend(s) of the given variable(s). Must be a cell array on the form: {'var1', 'Name', 'var2', 'Name2', ...}. Here we set the legends of the variables 'var1' and 'var2' to 'Name' and 'Name2' respectively. (The rest will be given the variable names as default).

To give a legend to a splitted line (second part) give;

```

{....,'Var1(second)','LegendSplitted',...} (Is case sensitiv)

Caution : legAuto should be set to 'on'. Which is not default.
          This is only important if the patch or fakeLegend
          property is used.

Caution : When given this property will overwrite the legends
          property

Caution : If the variable, scattergroup or candle is not found
          no error will occure!

Caution : If you have used the patch or fakeLegend property these
          can be translated as well with this property

```

Inherited from superclass NB_GRAPH

- legends ↑

Give the legends of the plot. Must be given as a cell array
of strings.

Default (it depends on the method you use) :

```

> 'graph'           : The variable names are used as the
                      default legends. (Given by the
                      variableToPlot and variableToPlotRight
                      properties)

> 'graphSubPlot'   : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)

> 'graphInfoStruct' : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)

```

Caution : The ordering of the legends is as follows:

- patch : See the patch property (only when legAuto is set to 'off')
- variablesToPlot : Then the all the variablesToPlot variables. When You have splitted lines, i.e. either using the dashedLine property or the lineStyles property, these can be set after all the variablesToPlot. Have in mind that the ordering is kept.
- variablesToPlotRight : Then the comes all the variables provided by the variablesToPlotRight. Then the splitted lines.
- fakeLegend : Then you can provide the legend text for the fake legend. (only when legAuto is set to 'off')

Example:

```

data      = nb_ts.rand('2012',10,2);
plotter = nb_graph_ts(data);
plotter.set('variablesToPlot',{'Var1'},...
            'variablesToPlotRight',{'Var2'},...
            'lineStyles',{'Var1',{'-','2014','--'},...
                         'Var2',{'-','2014','--'},...
            'legends',{'Test','Test (--)','Test2','Test2 (--)'});
plotter.graph()

```

Inherited from superclass NB_GRAPH

- **lineStyles** [↑](#)

Sets the line styles of the given variable(s). Must be a cell array on the form: {'var1', '-' , 'var2', '--', ...}. Here we set the line styles of the variables 'var1' and 'var2' to '-' and '--' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Supported line styles are '-','--',':','.-','---

Caution : For nb_graph_ts and nb_graph_data it is possible to split the plotted line using the following syntax;
{'var1',{'-','2000Q1','--'}} and {'var1',{'-',2,'--'}}
respectively

Inherited from superclass NB_GRAPH

- **lineWidth** [↑](#)

Sets the line width of the line plots. Must be a scalar.
Default is 2.5.

Inherited from superclass NB_GRAPH

- **lineWidths** [↑](#)

Sets the line widths of the given variables. Must be a cell array on the form: {'var1', 1, 'var2', 1.5,...}. Here we set the line widths of the variables 'var1' and 'var2' to 1 and 1.5 respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Inherited from superclass NB_GRAPH

- **localVariables** [↑](#)

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_GRAPH

- **lookUpMatrix** [↑](#)

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {  
    'mnemonics1','englishDescription1','norwegianDescription1';  
    'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_GRAPH

- **mYLim** ↑

Sets the max/min y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty this setting only sets the left y-axis limits.

Only an option for the graphInfoStruct(...) method.

- **markerSize** ↑

Sets the size of the all markers. Must be a scalar. Default is 8.

Inherited from superclass NB_GRAPH

- **markers** ↑

Sets the markers of the given variables. Must be a cell array on the form: {'var1', '^', 'var2', 'o', ...}. Here we set the markers of the variables 'var1' and 'var2' to '^' and 'o' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the table below on which markers that are supported.

'+'	: Plus sign
'o'	: Circle
'*'	: Asterisk
'.'	: Point
'x'	: Cross
's'	: Square
'd'	: Diamond
'^'	: Upward-pointing triangle
'v'	: Downward-pointing triangle
'>'	: Right-pointing triangle
'<'	: Left-pointing triangle
'p'	: Five-pointed star (pentagram)
'h'	: Six-pointed star (hexagram)

Inherited from superclass NB_GRAPH

- **missingValues** ↑

Set how to interpret missing observations. Must be a string. Either;

```
> 'interpolate' : Linearly interpolate the missing values.  
> 'none'          : No interpolation (default)
```

- **noLabel** ↑

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Both on the y-axis and x-axis. Default is 0. This is of course only have an effect if the nb_graph object have been given the xLabel or yLabel properties. (These are empty by default.)

Inherited from superclass NB_GRAPH

- **noLegend** ↑

Set it to 1 if no legend is wanted, otherwise set it to 0. Default is 0.

Inherited from superclass NB_GRAPH

- **noTickMarkLabels** ↑

Set to true (1) to remove tick marks and tick marks label of axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsLeft** ↑

Set to true (1) to remove tick marks and tick marks label of the left side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsRight** ↑

Set to true (1) to remove tick marks and tick marks label of the right side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the graphs, otherwise set it to 0. Default is 0.

Set it to 2 if you want the dataNames{ii} as the title of the graph when using the graph method.

Inherited from superclass NB_GRAPH

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_GRAPH

- **numberOfGraphs** ↑

Number of graph produced by the graph methods. Not settable.

Inherited from superclass NB_GRAPH

- **page** ↑

Sets the page of the data object to plot. Only an option for graph() method. Default is [], i.e. plot all pages in separate figures.

Inherited from superclass NB_GRAPH

- **parameters** ↑

A struct with the parameters that can be used in evaluation of expressions in the graphSubPlots and graphInfoStruct methods.

Inherited from superclass NB_GRAPH

- **patch** ↑

Sets the patch property of the object. If set this will add a patch (color fill) between two variables.

Only one patch :

```
{'patchName','var1','var2',color}
```

More patches :

```
{'patchName1','var1Patch1','var2Patch1',color1 ,...  
'patchName2','var1Patch2','var2Patch2',color2,...}
```

Where the the color option(s) must either be a 1 x 3 double with the RGB colors or a string with the color name.

The first input will be the legend text of the patch. I.e. 'patchName', 'patchName1' and 'patchName2' will be the description text included in the legend. (If not the property legAuto is set to 'off'.)

Inherited from superclass NB_GRAPH

- **patchAlpha** ↑

Sets the transparency of the patches. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_GRAPH

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_GRAPH

- **plotType** ↑

Sets the plot type. Either:

- > 'line' : If line plot(s) is wanted. Default.
- > 'stacked' : If stacked bar plot(s) is wanted. Not an option for the graphSubPlots(...) method.
- > 'grouped' : If grouped bar plot(s) is wanted.
- > 'dec' : If decomposition plot(s) is wanted. Which is a bar plot with also a line with the sum of the stacked bars. Not an option for the graphSubPlots(...) method.
- > 'area' : If area plot(s) is wanted.
- > 'candle' : If candle plot(s) is wanted.
- > 'scatter' : If scatter plot(s) is wanted.

- **plotTypes** ↑

Sets the plot type of the given variables. Must be a cell array on the form: {'var1', 'line', 'var2', 'grouped', ...}. Here we set the plot type of the variables 'var1' and 'var2' to 'line' and 'grouped' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the property plotType above on which plot types that are supported.

- **position** ↑

Sets the position of the axes, must be an 1x4 double. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

For the graph method graphSubPlots the position input can be given a 1 x N cell array, where each element is a 1x4 double.
N = obj.subPlotSize(1)*obj.subPlotSize(2).

Inherited from superclass NB_GRAPH

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

Inherited from superclass NB_GRAPH

- **scatterDates** ↑

Sets the start and end to be used for the scatter plot plotted against the left axes. Must be a nested cell array. The dates can be given as strings or date objects. E.g:

```
{'scatterGroup1',{ 'startDate1','endDate1'},...  
 'scatterGroup2',{ 'startDate2','endDate2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterDatesRight)

Be aware : Each date will result in one point in the scatter.

- **scatterDatesRight** ↑

Sets the start and end to be used for the scatter plot plotted against the right axes. Must be a nested cell array. The dates can be given as strings or date objects. E.g:

```
{'scatterGroup1',{ 'startDate1','endDate1'},...  
 'scatterGroup2',{ 'startDate2','endDate2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterDates)

Be aware : Each date will result in one point in the scatter.

- **scatterLineStyle** ↑

Sets the scatter line style. Must be string. See 'lineStyles' for supported inputs. (Sets the line style of all scatter plots)

- **scatterVariables** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the left y-axis. E.g:

```
{'var1','var2'}
```

- **scatterVariablesRight** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the right y-axis. E.g:

```
{'var1','var2'}
```

- **shading** ↑

The shading option of the axes background:

- 'grey' : Shaded grey background
- 'none' : Background color given by the color property.
- A n x m x 3 double with the background color. n is the number of horizontal pixels, m is the number of vertical pixels and 3 means the RGB colors.

Inherited from superclass NB_GRAPH

- **spacing** ↑

Sets the x-ticks spacing. Must be a scalar. Refers to the given x-axis tick frequency.

- **startGraph** ↑

Sets the start date of the graph. Must be a string or an object which of a subclass of the nb_date class. If this date is before the start date of the data, the data will be expanded with nan (blank values) for these periods.

- **subPlotSize** ↑

Sets the number of subplots per figure. Must be a 1 x 2 double with the number of subplot rows as the first element and the number of subplot columns as the second element. Only an option of the graphSubPlots() and graphInfoStruct() methods.

Default is [2, 2].

Inherited from superclass NB_GRAPH

- **subPlotSpecial** ↑

Set it to 1 if you want the 1x2 and 2x1 subplots to better fit for usage in presentations. Default is 0, i.e. use MATLAB default positions.

When set to 1 this class uses the nb_subplotSpecial instead of nb_subplot to find the positions of the subplots.

Inherited from superclass NB_GRAPH

- **subPlotsOptions** ↑

Use this property to set subplot spesific options of each variable while using the the method graphSubPlots(). Each field must be the name of the variable. The options are:

```
- yLabel      : The y-axis label.
- yLim       : Sets the y-axis limits. As a 1x2 double.
- ySpacing   : Spacing between y-axis tick marks.
- dashedLine : The date to start the dashed line for the given
               variable.
```

Example:

```
s.Var1 = struct('yLim',[0,1]);
s.Var2 = struct('yLim',[0,1]);
```

- **sumTo** ↑

Sets a number which the area or the stacked bar plot should sum to. Must be a scalar. I.e. if 100 is given. The sizes of the bars will be the percentage share of the total sum.

Inherited from superclass NB_GRAPH

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_GRAPH

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_GRAPH

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_GRAPH

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_GRAPH

- **variablesToPlot** ↑

Sets the variables to plot against the left (or both) axes. Must be a cell array of strings with the variable names. I.e. {'var1', 'var2',...}.

Not an option for the graphInfoStruct(...), which use the property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **variablesToPlotRight** ↑

Sets the variables to plot against the right axes. Must be a cell array of strings with the variable names. I.e. {'var1', 'var2',...}.

Not an option for the graphInfoStruct(...), which use the property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **verticalLine** ↑

Sets the date(s) of where to place a vertical line(s). (Dotted orange line is default). Must be a string or a cell array of strings with the date(s) for where to place the vertical line(s). I.e. 'date1' or {'date1','date2',...}

Each element of the cell could also be given as a cell with two dates, i.e. {'date1', 'date2' }. Then the vertical line will be placed between the given dates.

If plotType is set to 'scatter' scalars can be used instead.

- **verticalLineColor** ↑

Sets the color(s) of the given vertical line(s). Must either be an $M \times 3$ double with the RGB color(s) or a $1 \times M$ cell array of strings with the color name(s). Where M must be less than the number of plotted vertical lines.

If less or no color(s) is/are set by this property the default color(s) for the rest or all the vertical line(s) is/are [0 0 0]. I.e. MATLAB black.

- **verticalLineLimit** ↑

Sets the y-axis limit(s) of the given vertical line(s). Must either be a $1 \times M$ cell array of 1×2 double with the upper and lower y-axis limit(s). Where M must be less than the number of plotted vertical lines.

If less or no limit(s) is/are set by this property the default limit for the rest or all the vertical line(s) is/are the full height of the graph.

- **verticalLineStyle** ↑

Sets the line style(s) of the given vertical line(s). Must either be a $1 \times M$ cell array of strings with the style(s). Where M must be less than the number of plotted vertical lines.

If less or no style(s) is/are set by this property the default line style is/are used for the rest or all the vertical line(s) is/are '--'.

- **verticalLineWidth** ↑

Sets the line width of (all) the vertical line(s). Must be a scalar. Default is 1.5.

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar. Default is 12.

Inherited from superclass NB_GRAPH

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel
property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' |
'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **xLim** ↑

Sets the x-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Both or one of them can be
set to nan, i.e. the default limits will be used) E.g. [0 6],
[nan, 6] or [0, nan].

Only an option when the plotType property is set to 'scatter'.

- **xScale** ↑

Sets the scale of the x-axis. Either {'linear'} | 'log'.

Only an option when plotType is set to 'scatter'.

- **xTickLabelAlignment** ↑

The alignment of the x-axis tick marks labels. {'normal'} |
'middle'.

- **xTickLabelLocation** ↑

The location of the x-axis tick marks labels. Either
{'bottom'} | 'top' | 'baseline'. 'baseline' will only work in
combination with the xtickLocation property set to a double
with the basevalue.

- **xTickLabels** ↑

Sets the x-axis tick mark labels. To translate the default tick
mark label 'Var1' and 'Var2' you can use;

```
{'Var1','Label1','Var2','Label2'}
```

If given as a empty cell, default x-axis tick marks will be used, which is default.

This property can be used to create multi-lined x-axis tick mark labels. To do this give a multi-row char to the labels you want to make multi-lined.

Inherited from superclass NB_GRAPH

- **xTickLocation** ↑

The location of the x-axis tick marks. Either {'bottom'} | 'top' | double. If given as a double the tick marks will be placed at this value (where the number represent the y-axis value).

- **xTickRotation** ↑

Sets the rotation of the x-axis tick mark labels. Must be a scalar with the number of degrees the labels should be rotated. Positive angles cause counterclockwise rotation.

- **yDir** ↑

Sets the direction of the y-axis of the plot. Must be a string. Either 'normal' or 'reverse'. Default is 'normal'.

Caution: If the variablesToPlotRight is not empty this setting only sets the left y-axis direction.

- **yDirRight** ↑

Sets the direction of the right y-axis of the plot. Either 'normal' or 'reverse'. Default is 'normal'. (Only when the variablesToPlotRight property is not empty.)

- **yLabel** ↑

Sets (add) the text of the y-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **yLabelFontSize** ↑

Sets the font size of the y-axis label. Must be scalar. Default is 12.

Inherited from superclass NB_GRAPH

- **yLabelFontWeight** ↑

Sets the font weight of the y-axis label. Must be string. Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **yLabelInterpreter** ↑

The interpreter used for the given string given by the `xLabel` property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass `NB_GRAPH`

- **yLabelRight** ↑

Sets the text of the y-axis label. (Only on the right side.)
Must be a string. Takes the same font option as the `yLabel` property

Inherited from superclass `NB_GRAPH`

- **yLim** ↑

Sets the y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Caution: If the `variablesToPlotRight` property is not empty this setting only sets the left y-axis limits.

Only an option for the `graph(...)` and `graphSubPlots()` methods.
(For the `graphInfoStruct(...)` method use the property `GraphStruct`.)

- **yLimRight** ↑

Sets the right y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Only when the `variablesToPlotRight` property is not empty.) (Both or one of them can be set to nan, i.e. the default limits will be used)
E.g. [0 6], [nan, 6] or [0, nan].

Only an option for the `graph(...)` and `graphSubPlots()` methods.
(For the `graphInfoStruct(...)` method use the property `GraphStruct`.)

- **yOffset** ↑

The y-axis tick mark labels offset from the axes

- **yScale** ↑

Sets the scale of the y-axis. {'linear'} | 'log'. (Both the left and the right axes if nothing is plotted on the right axes.)

- **yScaleRight** ↑

Sets the scale of the right y-axis. {'linear'} | 'log'.
(Has only something to say if something is plotted on the
right axes)

- **ySpacing** ↑

Sets the spacing between y-axis tick mark labels of the
y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the
tick marks labels. The number of periods will follow
the frequency of the xTickFrequency property if
setted, else it will follow the frequency of the
data provided. (Except when dealing with daily data,
because then the default xTickFrequency is 'yearly')

Caution: If the variablesToPlotRight property is not empty
this property only sets the spacing between ticks of
the left y-axis.

- **ySpacingRight** ↑

Sets the spacing between y-axis tick mark labels of the
right y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the
tick marks labels. The number of periods will follow

Methods:

- | | | | |
|-----------------------------------|-------------------------------------|---------------------------------------|------------------------------|
| • createFigure | • get | • getBDOOptions | • getCode |
| • getData | • getGenericOptions | • getPlottedVariables | • graph |
| • graphInfoStruct | • graphSubPlots | • isempty | • merge |
| • realTimespan | • resetDataSource | • saveData | • saveFigure |
| • set | • struct | • timespan | • update |

► **createFigure** ↑

`createFigure(obj)`

Description:

Create figure to plot in.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_graph_bd

Input:

- obj : An object of class nb_graph_bd
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_graph_ts object. Be aware that all the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_graph_bd object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#), [nb_bd](#)

Written by Kenneth S. Paulsen

► **getBDOptions ↑**

```
options = getBDOptions(obj)
```

Description:

Container with the properties specific to nb_graph_bd

Input:

- none

Output:

- options : A numSettings name 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getBDOptions(obj);
```

See also:

[set](#), [getGenericOptions](#), [nb_parseInputs](#)

Written by Kenneth Sæterhagen Paulsen

► **getCode** ↑

```
[code,numOfGraphs] = getCode(obj,frameTitle,frameSubTitle,...  
                               footer,source,pageNumber,theme)
```

Description:

Get the latex code when including a graph in a slide object produced by an object of class nb_graph_ts.

This method is called from the nb_Slide class, when making a beamer slide in latex.

Input:

- obj : An object of class nb_graph_ts
- frameTitle : A string with the frame title of the slide.
- frameSubTitle : A string with the frame subtitle of the slide.
- footer : A cellstr with the footers of the slide
- source : A cellstr with the sources of the slide
- pageNumber : Give 1 if the page number should be included.
- theme : The beamer theme used. Default is '' (use default beamer theme).

Output:

- code : The latex code as a char.
- numOfGraphs : The number of graphs produced by the input object of class nb_graph_ts. As an integer.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **getData** ↑

```
data = getData(obj)
```

Description:

Get the data of the graph

Input:

- obj : An object of class nb_graph_bd

Output:

- data : As an nb_bd object.

Written by Kenneth S. Paulsen

► **getGenericOptions** ↑

```
options = getGenericOptions(obj)
```

Description:

A static method of nb_graph.

Container with the intersect of options for subclasses of nb_graph.

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getGenericOptions(obj);
```

See also:

[set](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj,forLegend)
```

Description:

Get the plotted variables of the graph

Input:

- obj : An object of class nb_graph_bd
- forLegend : true or false (default)

When forLegend is used the variables are listed as needed for the legend, but without fake legends and patches, and duplicates are not removes.

Output:

- vars : As a cellstr

Example:

```
vars = obj.getPlottedVariables();
```

Written by Kenneth S. Paulsen

► **graph** ↑

```
graph(obj)
```

Description:

Create a graph or more graphs

Graphs all the variables given by the variablesToPlot and the variablesToPlotRight properties in one plot, and does that for all the datasets of the DB property. I.e. one new plot for each dataset (page) of the DB property of the nb_graph_ts object.

Input:

- obj : An object of class nb_graph_bd

Output:

The graph plotted on the screen.

Examples:

```
data = nb_bd([2;1;2],'', '2012Q1', 'Var1');
obj = nb_graph_bd(data);
obj.graph();
```

See also:

[nb_graph_bd.set](#)

Written by Kenneth S. Paulsen

► **graphInfoStruct** ↑

`graphInfoStruct(obj)`

Description:

Create graphs based on the property `GraphStruct`. (Or the default `GraphStruct` given by the method `defaultGraphStruct`)

This method makes it easy to set up graph packages. For example graphing output from models.

See the documentation NB toolbox for more on this method.

Input:

- `obj` : An object of class `nb_graph_bd`, where the property `GraphStruct` contains the information on how to plot the objects data.

Output:

The wanted graphs plotted on the screen.

Written by Kenneth S. Paulsen

► **graphSubPlots** ↑

`graphSubPlots(obj)`

Description:

Create graphs of all the variables given in `variablesToPlot` property, in different subplots. (Each dataset against each other). The number of subplot is given by the property `subPlotSize`.

Input:

- obj : An object of class nb_graph_bd

Output:

The wanted graphs plotted on the screen.

Written by Kenneth S. Paulsen

► isempty ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_graph_bd object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_graph_bd

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► merge ↑

```
out = merge(obj,varargin)
```

Description:

Default: This tries to merge the data of plotter objects that uses the graphSubPlots, or the graphInfoStruct.

Caution: All objects must share the exact same variables!

'graph' is given: In this case it will try to merge the data of all nb_graph objects, and concatenate the variableToPlot property. If the property variableToPlot is empty for all objects, it will also be returned as empty, i.e. plot all series in the data.

Caution : Merging of any of the other properties then DB and variableToPlot is not done!

Input:

- obj : An object of class nb_graph, which includes the classes nb_graph_ts, nb_graph_cs and nb_graph_data.

Optional input:

- 'graph' : Give this input to merge the graphs in the way that is described in the description of this method.
- varargin : Each input must be an object of class nb_graph.

Output:

- out : An object of class nb_graph.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► **realTimespan** ↑

```
dateString = realTimespan(obj,language,freq)
```

Description:

Get the timespan of the data behind a graph which is not nan or infinite, as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_bd
- language : 'english' or 'norwegian'. 'norwegian' is default
- freq : The wanted frequency of the timespan

Output :

- dateString : A string with the timespan of the graph represented by this object.

Written by Kenneth S. Paulsen

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_graph_bd object.

Input:

- obj : An object of class nb_graph_bd
- dataSource : An nb_bd object with the new data.
- updateProps : If the variablesToPlot should be updated to be equal to dataSource.variables + the startGraph and endGraph properties are set to the startDate and endDate of the new data source.

Can also be 'gui'. Then all properties will be checked against the new dataset, and if the new dataset does not comply with the properties, the properties will be updated!

Written by Kenneth S. Paulsen

► **saveData** ↑

```
obj = saveData(obj,filename)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_graph_bd
- filename : A string with the saved output name

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to (a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_graph
- extraName : If you want to add some extra name to the saveName property of the nb_graph_data object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Examples:

```
data = nb_data([2;1;2],'', '1', 'Var1');
obj = nb_graph_data(data);
obj.graph();
saveFigure(obj,'test','pdf');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► set ↑

```
set(obj,varargin)
```

Description:

Set properties of the nb_graph class and its subclasses.

See documentation NB Toolbox or type help('nb_graph') for more on the properties of the class.

Input:

- obj : An object that is a subclass of nb_graph
- varargin : 'propertyName',PropertyValue,...

Output:

No actual output, but the input objects properties will have been set to their new value.

Examples:

```
data = nb_ts([2;1;2],'', '2012Q1', 'Var1');
obj = nb_graph_ts(data);
obj.set('startGraph', '2012Q1', 'endGraph', '2012Q3');
obj.set('crop', 1);
```

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct

Input:

- obj : An object of class nb_graph_ts

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **timespan** ↑

```
dateString = timespan(obj,language,freq)
```

Description:

Get the timespan of the graph as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_bd
- language : 'english' or 'norwegian'. 'norwegian' is default
- freq : The wanted frequency of the timespan

Output :

- dateString : A string with the timespan of the graph represented by this object.

Written by Kenneth S. Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the figure

See the update method of the nb_ts classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_graph_bd

Output:

- obj : An object of class nb_graph_bd, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ **nb_graph_cs**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_cs(varargin)
```

Superclasses:

handle, nb_graph

Description:

This is a class for making graphics of cross sectional data.

Constructor:

```
obj = nb_graph_cs(data)
```

Constructor of the nb_graph_cs class

Input:

- data : If you have one source you want to collect the data from you can give it as a input to this constructor. I.e:

```
> nb_cs           : nb_graph_cs(data)
> excel          : nb_graph_cs('excelName1')
```

It is also possible to initialize an empty nb_graph_cs object. (Do not provide any inputs to the constructor)

Output:

- obj : An object of class nb_graph_cs

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_adv](#), [nb_graph_subplot](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- DB
- a4Portrait
- addSpace
- annotation
- areaAccumulate
- axesFast
- axesFontSizeX
- axesLineWidth
- axesScaleFactor
- barAlpha1
- barBlend
- barOrientation
- barShadingDirection
- barWidth
- baseLineColor
- baseLineWidth
- candleIndicatorColor
- candleMarker
- candleWidth
- colorMap
- colorOrderRight
- crop
- donutInnerRadius
- excelFooterEng
- excelTitleEng
- factor
- fanColor
- GraphStruct
- addAdvanced
- alignAxes
- areaAbrupt
- areaAlpha
- axesFontSize
- axesFontWeight
- axesPrecision
- axesScaleLineWidth
- barAlpha2
- barLineWidth
- barShadingColor
- barShadingTypes
- baseLine
- baseLineStyle
- baseValue
- candleIndicatorLineStyle
- candleVariables
- code
- colorOrder
- colors
- displayValue
- donutRadius
- excelFooterNor
- excelTitleNor
- fakeLegend
- fanDatasets

- fanLegend
- fanLegendFontSize
- fanLegendLocation
- fanLegendText
- fanMethod
- fanPercentiles
- fanVariable
- figureColor
- figureName
- figurePosition
- figureTitle
- fileFormat
- findAxisLimitMethod
- flip
- fontName
- fontScale
- fontUnits
- graphStyle
- grid
- gridLineStyle
- highlight
- highlightColor
- horizontalLine
- horizontalLineColor
- horizontalLineStyle
- horizontalLineWidth
- language
- legAuto
- legBox
- legColor
- legColumnWidth
- legColumns
- legFontColor
- legFontSize
- legInterpreter
- legLocation
- legPosition
- legReorder
- legSpace
- legendText
- legends
- lineStyles
- lineWidth
- lineWidths
- localVariables
- lookUpMatrix
- markerSize
- markers
- noLabel
- noLegend
- noTickMarkLabels
- noTickMarkLabelsLeft
- noTickMarkLabelsRight
- noTitle

- normalized
- page
- patch
- pdfBook
- pieEdgeColor
- pieLabelsExtension
- pieTextExplode
- plotType
- position
- radarNumberOfIsoLines
- radarScale
- scatterLineStyle
- scatterTypesRight
- scatterVariablesRight
- subPlotSize
- sumTo
- title
- titleFontSize
- titleInterpreter
- typesToPlot
- userData
- variablesToPlotRight
- verticalLineColor
- verticalLineStyle
- xLabel
- xLabelFontSize
- xLabelInterpreter
- numberOfGraphs
- parameters
- patchAlpha
- pieAxisVisible
- pieExplode
- pieOrigPosition
- plotAspectRatio
- plotTypes
- radarFontColor
- radarRotate
- saveName
- scatterTypes
- scatterVariables
- shading
- subPlotSpecial
- tag
- titleAlignment
- titleFontWeight
- titlePlacement
- typesToPlotRight
- variablesToPlot
- verticalLine
- verticalLineLimit
- verticalLineWidth
- xLabelAlignment
- xLabelFontWeight
- xLabelPlacement

- `xLim`
- `xScale`
- `xSpacing`
- `xTickLabelAlignment`
- `xTickLabelLocation`
- `xTickLabels`
- `xTickLocation`
- `xTickRotation`
- `yDir`
- `yDirRight`
- `yLabel`
- `yLabelFontSize`
- `yLabelFontWeight`
- `yLabelInterpreter`
- `yLabelRight`
- `yLim`
- `yLimRight`
- `yOffset`
- `yScale`
- `yScaleRight`
- `ySpacing`
- `ySpacingRight`

- **`DB`** 

All the data given to the `nb_graph_cs` object collected in a `nb_cs` object. Should not be set using `obj.DB = data!` Instead see the `nb_graph_cs.resetDataSource` method.

- **`GraphStruct`** 

Sets the information on which variables and how to plot them when using the `graphInfoStruct(...)` method of this class.

Must be an .m-file with the code on how the structure of graphing information should be initialized or as a structure with the graphing information.

Only the `graphInfoStruct(...)` method uses this property. See the documentation of the NB toolbox for more on this input.

Inherited from superclass `NB_GRAPH`

- **`a4Portrait`** 

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass `NB_GRAPH`

- **`addAdvanced`** 

Set to false to prevent adding advanced components.

Inherited from superclass `NB_GRAPH`

- **`addSpace`** 

Sets the space before and/or after in the x direction of the graph [`addedSpaceBefore` `addedSpaceAfter`]. E.g. add one period before and after [1,1].

Inherited from superclass `NB_GRAPH`

- **alignAxes** ↑

Align axes at a given base value. Must be set to a scalar double.
E.g. 0.

Inherited from superclass NB_GRAPH

- **annotation** ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the nb_annotation class for more. If you give more objects they must be collected in a cell array.

Examples can be found here:

\NBTOOLBOX\Examples\Graphics\nb_annotationExamples.m

Inherited from superclass NB_GRAPH

- **areaAbrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan. Default is false.

Inherited from superclass NB_GRAPH

- **areaAccumulate** ↑

Set if the areas should be accumulated or not. Default is true.

Inherited from superclass NB_GRAPH

- **areaAlpha** ↑

Sets the transparency of the area chart. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **axesFast** ↑

Set the if you want graphs to be produced fast with a loss of functionality and look or not. When empty default for the graph method is false, while for the graphSubPlots and graphInfoStruct methods are true.

Inherited from superclass NB_GRAPH

- **axesFontSize** ↑

Sets the font size of the axes tick mark labels, default is 12. Must be a scalar.

Inherited from superclass NB_GRAPH

- **axesFontSizeX** ↑

Sets the font size of the x-axis tick mark labels, default is []. I.e. use the axesFontSize property instead. Must be a scalar.

This property will not be scaled when fontUnits are changed, as is the case for axesFontSize.

Caution: Will not set the x-axis font size when plotType is set to scatter.

Inherited from superclass NB_GRAPH

- **axesFontWeight** ↑

Sets the font weight of the axes tick mark labels. Must be a string. Default is 'normal'. Either 'normal', 'bold', 'light' or 'demi'

Inherited from superclass NB_GRAPH

- **axesLineWidth** ↑

Sets the line width of the axes, default is 0.5. Must be a set to a scalar double greater than 0.

Inherited from superclass NB_GRAPH

- **axesPrecision** ↑

Sets the precision/format of the rounding of number on the axes.

See the precision input to the nb_num2str function. Default is [], i.e. to call num2str without additional inputs.

Inherited from superclass NB_GRAPH

- **axesScaleFactor** ↑

Set the scaling factor used when axesScaleLineWidth is set to true.

Inherited from superclass NB_GRAPH

- **axesScaleLineWidth** ↑

Scale line width to axes height. true or false (default). If axesScaleFactor is set to a scalar number this will be the scaling factor used instead of the automatic scaling factor.

Inherited from superclass NB_GRAPH

- **barAlpha1** ↑

Set the alpha blending parameter 1, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barAlpha2** ↑

Set the alpha blending parameter 2, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barBlend** ↑

Set to true to do alpha blending instead of shading if shading options is used. Default is false.

Inherited from superclass NB_GRAPH

- **barLineWidth** ↑

Set the bar line width without affecting any other line widths. Default is empty, i.e. use the lineWidth property.

Inherited from superclass NB_GRAPH

- **barOrientation** ↑

Sets the orientation of the bar plot. This property is discarded when using the plotTypes property. Must be a string with either 'vertical' (default) or 'horizontal'

- **barShadingColor** ↑

The color shaded bars are interpolated with. Default is 'white'. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **barShadingDirection** ↑

Sets the angle the shading should be vertical of. Either {'north'} | 'south' | 'west' | 'east'. As a string. Default is 'north'.

Inherited from superclass NB_GRAPH

- **barShadingTypes** ↑

Sets the types which should have shaded bars. Must be a cellstr, with the names of the types to be shaded. E.g. {'Type1','Type2',...}

- **barWidth** ↑

Sets the width of the bar plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **baseLine** ↑

If 0 is given the base line will not be plotted, otherwise it will be plotted.

Inherited from superclass NB_GRAPH

- **baseLineColor** ↑

Sets the color of the base line. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **baseLineStyle** ↑

Sets the line style of the base line. As a string.
Either {'-' | '--' | '---' | ':' | '-.' | 'none'.

Inherited from superclass NB_GRAPH

- **baseLineWidth** ↑

Sets the width of the base line. Must be a scalar.

Inherited from superclass NB_GRAPH

- **baseValue** ↑

Sets the base value used for bar and area plot. (Sest also the
y-axis base value for the base line)

Inherited from superclass NB_GRAPH

- **candleIndicatorColor** ↑

Sets the color of the indicator of the candle. Must either be a
1 x 3 double with the RGB colors or a string with the color
name.

Inherited from superclass NB_GRAPH

- **candleIndicatorLineStyle** ↑

Sets the line style of the indicator of the candle. Must be a
string. Default is '-'.

Inherited from superclass NB_GRAPH

- **candleMarker** ↑

Sets the marker of the indicator of the candle. Must be a
string. Default is 'none'.

Inherited from superclass NB_GRAPH

- **candleVariables** ↑

Sets which variables are going to be plotted as open, close,
high, low and indicator. Must be a cellstr. E.g.

{'open','var1','close','var2',...}

If a type (e.g. 'low') is not given it will not be plotted.

Meaning of each type:

> 'close' : The lowest values of the plotted patches of
the candle.

> 'high' : The highest values of the plotted candles.

> 'indicator' : The value for where to plot a horizontal line
(across the patch). E.g. the mean value.

> 'open' : The highest values of the plotted patches of the candle.

> 'low' : The lowest values of the plotted candles.

Inherited from superclass NB_GRAPH

- **candleWidth** [↑](#)

Sets the width of the candle plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **code** [↑](#)

If you set this to a file name, this file will be evaluated after all the plotting is done. Must be a string.

This could be nice for adding special annotation, lines and so on. In this file you can use any MATLAB function, but some strange error message can occur if you use local variables which is already defined (So use long and descriptive variable names in the provided code to prevent this).

This property is only an option for the method graph.

Nice to know:

> The current nb_figure object can be reached with
get(obj,'figureHandle')

> The current nb_axes object can be reached with
get(obj,'axesHandle')

Caution : Be aware that the name of the file must be unique and cannot be a MATLAB function or a local variable.

Caution : The file must be a MATLAB .m file

Inherited from superclass NB_GRAPH

- **colorMap** [↑](#)

Sets the colormap used when plotType is set to 'image' and fanMethod is set to 'graded'. Must be given as n x 3 double or the path to a .mat file that contain the colormap. Default is given by nb_axes.defaultColorMap.

For an example of a supported MAT file see:

- ...\\Examples\\Graphics\\colorMapNB.mat

Inherited from superclass NB_GRAPH

- **colorOrder** [↑](#)

Sets the color order of the plotted data (Left axes). Either a cellstr with the color names (size; 1xM) or a double with

RGB colors (Size; Mx3). Where M is the number of plotted variables against the left axes.

Inherited from superclass NB_GRAPH

- **colorOrderRight** [↑](#)

Sets the color order of the plotted data (Right axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the right axes.

Inherited from superclass NB_GRAPH

- **colors** [↑](#)

Sets the colors of the given variable(s). Must be a cell array on the form: {'var1', 'black', 'var2', [0.2 0.2 0.2],...}. Here we set the colors of the variables 'var1' and 'var2' to 'black' and [0.2 0.2 0.2] (RGB color) respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Caution: For the graphInfoStruct() method you need to select the variables from obj.DB.dataNames instead!

Inherited from superclass NB_GRAPH

- **crop** [↑](#)

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_GRAPH

- **displayValue** [↑](#)

Set to false to not display observation values when holding the mouse over the graph. Default is true.

Inherited from superclass NB_GRAPH

- **donutInnerRadius** [↑](#)

Sets the realtive length of the inner circle of the donut. -0.5 mean that the donut has an inner circle with the same radius as half the outer circle of the donut.

- **donutRadius** [↑](#)

Sets the radius of the donut. Default is 1.

- **excelFooterEng** [↑](#)

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. English version.

Inherited from superclass NB_GRAPH

- **excelFooterNor** ↑

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. Norwegian version.

Inherited from superclass NB_GRAPH

- **excelTitleEng** ↑

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). English version.

Inherited from superclass NB_GRAPH

- **excelTitleNor** ↑

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). Norwegian version.

Inherited from superclass NB_GRAPH

- **factor** ↑

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_GRAPH

- **fakeLegend** ↑

Set this property if you want to add a legend of none plotted variable. You can set this property. Must be a cell on the form: {'legendName', settings,...}, where the legendName will be text of the fake legend. And the settings input must also be a cell array with the optional inputs {...,'propertyName', propertyName,...}. These settings will set the appearance of the fake legend. The following propertyNames are supported:

```
> 'type'      : Either 'line' or 'patch'.  
  
> 'color'     : Sets the color of the fake legend. Default is  
  'black'. Must either be a 1 x 3 double with  
  the RGB colors or a string with the color  
  name. If shaded patch is wanted it must be  
  2 x 3 double with the RGB colors or a 1 x 2  
  cellstr with the color names to use.  
  
> 'direction' : Sets the direction of the shading. Either  
  {'north'} | 'south' | 'west' | 'east'. As a  
  string. Default is 'north'. Only an option  
  when 'type' is set to 'patch'.  
  
> 'edgeColor' : Sets the color of the edge of the patch.  
  Default is the same as the 'color' option.
```

Only an option when 'type' is set to 'patch'.

> 'lineStyle' : Sets the line style of the fake legend. As a string. Either {'-' | '--' | '---' | ':' | '-.' | 'none'. Default is a normal line.

Will set the edge line style if the 'type' is set to 'patch'.

> 'lineWidth' : Sets the line width of the fake legend. Default is 2.5. Must be a scalar.

> 'marker' : Sets the marker of the fake legend. Default is 'none'. Only an option when 'type' is set to 'line'. Lookup LineSpec in the MATLAB help meny for more on the supported marker types.

Caution : Even if you don't want to set any of the optional input. The settings input must be given. I.e an empty cell, i.e. {}.

Inherited from superclass NB_GRAPH

- **fanColor** [↑](#)

Sets the colors of the fan charts. Must be a matrix of size; number of fan layer x 3, with the RGB color specifications.

or

A string with the basis color for the fan colors. Either 'red', 'blue', 'green', 'yellow', 'magenta', 'cyan' or 'white'.

Be aware that the classification of the color of the given "basecolor" is based on the property fanPercentiles, so if you have more or less than 4 layers you must change this property to have the same size as the number of layers. (This is the case even if you don't use it to calculate the percentiles of a given datasets. Given by the property fanDatasets)

Default is the nb colors.

Caution: If fanMethod is set to 'graded' the color used are those assign to the colorMap property.

- **fanDatasets** [↑](#)

Sets the data of the added fan layers.

If you have the data of the different layer in separate datasets. Then the input to this property must be a 2 * number of layers cell matrix. Can be excel spreadsheets or .mat files.

or

You can also give the data with all the simulated data or fan layers in a nb_cs object. (Each page one simulation or each

page one layer).

Remember that the percentiles for the fan charts is set by the property `fanPercentiles`, default is `[.3, .5, .7, .9]`.

The fan layers are added differently given the method you use.

```
> graph(...): Adds the fan given in this property to the last
   variable given in the property variablesToPlot.
   (If not the fanVariable property is set.) If you
   graph more datasets (in separate figures), the
   fan layers will be the same for all of them.
   Set the fanVariables property if that is not
   wanted.

> graphSubPlots(...): Adds the fan given in this property to
   the corresponding variable in the last
   given datasets (last page of the nb_cs
   object given by the property DB) of each
   each subplot.

-graphInfoStruct(...): Adds the fan given by this property to
   each variable of each subplot. Added
   for the last given dataset. (Last page
   of the nb_cs object given by the
   property DB.)
```

- **fanLegend** [↑](#)

Set this property if you want to add a MPR styled legend of the fan layers. Set it to 1 if wanted. Default is not (0).

- **fanLegendFontSize** [↑](#)

Sets the font size of the fan legend. Default is 12. Must be a scalar.

- **fanLegendLocation** [↑](#)

Sets the location of the fan legend. Locations are:

```
> 'Best', (same as 'North'.)
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'outside' : Places the legend outside the axes, one the
   right side. (If you create subplots, this is
   the only option which will look good.)
```

You can also give a 1×2 double vector with the location of where to place the legend. First column is the x-axis location and the second column is the y-axis location. Both must be between 0 and 1.

- **fanLegendText** ↑

Set this property if you want to give the text over the colored boxes of the MPR looking fan legend, you must give them as a cell. (Must have same size as number of layers.) Default is the percentiles with the % sign added. E.g. {'30%', '50%', '70%', '90%'}.

- **fanMethod** ↑

{'percentiles' | 'hdi' | 'graded'}; Which type of fan chart is going to be made. 'percentiles' are produced by using percentiles, 'hdi' use highest density intervals, while 'graded' creates a graded fan chart of the colors specified with the colorMap property.

- **fanPercentiles** ↑

Sets the percentiles you want to calculate (if the fanDatasets property is representing simulation) or specify the percentiles you have given (if the fanDatasets property is representing already calculated percentiles). Must be given as a double vector. Default is [0.3 0.5 0.7 0.9].

This property is also the default base for the text of the fan legend. (Where the percentiles are given in percent.)

- **fanVariable** ↑

Sets the variable you want to add the fan layers to. Must be given as a string with the variable name. Default is to use the last variable in the variablesToPlot property.

- **figureColor** ↑

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_GRAPH

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_GRAPH

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_GRAPH

- **figureTitle** ↑

Sets if figure titles is wanted on the figures when using the method graphInfoStruct. The fieldnames of the GraphStruct property will be used as figure title. Default is 0 (not include).

If set to a one line char this property can be used for the plotting method graphSubPlots.

Inherited from superclass NB_GRAPH

- **fileFormat** [↑](#)

Sets the save file format. Either 'eps', 'jpg', 'pdf' or 'png'. 'pdf' is default.

Inherited from superclass NB_GRAPH

- **findAxisLimitMethod** [↑](#)

Sets the method used for finding the axis limits, and y-ticks.

- > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
- > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
- > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
- > 4 : Uses the MATLAB algorithm for finding the axis limits.

Inherited from superclass NB_GRAPH

- **flip** [↑](#)

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_GRAPH

- **fontName** [↑](#)

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_GRAPH

- **fontScale** [↑](#)

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_GRAPH

- **fontUnits** [↑](#)

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set

before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : Don't use this property in combination with one of the graph styles defined by this class. I.e. 'mpr', 'mpr_white', 'presentation' and 'presentation_white'. They will set this property to 'normalized'.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_GRAPH

- **graphStyle** [↑](#)

Sets the graphing style of the plots. Give 'mpr' if you want the MPR looking style, or 'presentation' for graphs for the presentation (see nb_Presentation). If you don't want the grey shaded background you can use 'mpr_white' and 'presentation_white' instead.

You can also add your own graph style setting through a .m file. I.e. give the filename (without extension) as a string.

E.g. In the file you can type in something like:

```
set(obj,'titleFontSize',12,'legInterpreter','tex');
```

Caution: If you give this as the first input to the set method it is possible to overrun some of the default settings if that is wanted. E.g. font size of the text, the shading of the plot and so on.

Inherited from superclass NB_GRAPH

- **grid** [↑](#)

Set it to 'on' if grid lines are wanted otherwise set it to 'off'. Default is 'off'.

Inherited from superclass NB_GRAPH

- **gridLineStyle** [↑](#)

Sets the line style of the grid lines. Either '-', '--', ':' or '-.'.

Inherited from superclass NB_GRAPH

- **highlight** [↑](#)

If highlighted area between two types of the graph is wanted set this property.

One highlighted area:

A cell array on the form {'type1', 'type2'}

More highlighted areas:

A nested cell array on the form

{{'type1(1)', 'type2(1)'}, {'type1(2)', 'type2(2)'}, ...}

If plotType is set to 'scatter' scalars can be used instead.

- **highlightColor** ↑

Sets the colors of the background patches. Must be one of the following:

- > A string with the color name to use when plotting (all) the patch(es)
- > A cell array of strings with the color names
- > A cell with 1 x 3 doubles with the RGB color specification.

When given as a cell this property must match the highlight property. I.e. the number of highlighted areas added.

The default color of the highlighted areas is 'light blue'.

- **horizontalLine** ↑

If you want to include some extra horizontal lines, beside the base line, you can set this property. Must be set to a scalar or a double vector with the y-axis value(s) on where to place the horizontal line(s).

Inherited from superclass NB_GRAPH

- **horizontalLineColor** ↑

Sets the color(s) of the given horizontal line(s).

Must either be an M x 3 double with the RGB color(s) or a 1 x M cell array of strings with the color name(s). Where M must be less than the number of plotted horizontal lines.

If less or no color(s) is/are set by this property the default colors for the rest or all the horizontal line(s) is/are [0 0 0], i.e. MATLAB black.

Inherited from superclass NB_GRAPH

- **horizontalLineStyle** ↑

Sets the style(s) of the given horizontal line(s).

Must either be a string or a 1 x M cell array of strings with the style(s). Where M must be less than the number of plotted horizontal lines.

If less or no style(s) is/are set by this property the default lines for the rest or all the horizontal line(s) is/are '-'.

Inherited from superclass NB_GRAPH

- **horizontalLineWidth** ↑

Sets the line width of the horizontal line(s). Must be a scalar. Default is 1.

Inherited from superclass NB_GRAPH

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_GRAPH

- **legAuto** ↑

'on' | {'off'}. As a string.

> 'on' : The strings given through the fakeLegend and patch properties are automatically added to the legend.

> 'off' : The strings given through the fakeLegend and patch properties are not automatically added to the legend. Which means that you must provide all the legend information through the legends property. The patch descriptions must be given first and the fake legend description must be given last in the legends property. Default.

Inherited from superclass NB_GRAPH

- **legBox** ↑

Set if a box should be drawn around the legend. Either {'on'} | 'off'.

Caution : When removing the box you make it impossible to move it around afterwards.

Inherited from superclass NB_GRAPH

- **legColor** ↑

Sets the color of the background of the box of the legend. Either as 1x3 double with the RGB or as a string). Default is 'none', i.e. transparent.

Inherited from superclass NB_GRAPH

- **legColumnWidth** ↑

Sets the width of the columns of the legend. Default is to let MATLAB find the width itself. If given it must be a scalar with the width of all columns of the legend or a 1 x legColumns double with the width of each individual column.

Should be between 0 and 1. Use the try and fail method to find out what fits.

Inherited from superclass NB_GRAPH

- **legColumns** ↑

Sets the number of columns of the legends. Must be a scalar.

Inherited from superclass NB_GRAPH

- **legFontColor** ↑

Sets the font color(s) of the legend text. Must either be a 1x3 double or a string with the color name of all legend text objects, or a M x 3 double or a cellstr array with size 1 x M with color names to use of each legend text object.

Inherited from superclass NB_GRAPH

- **legFontSize** ↑

Sets the font size of the legend. Must be a scalar default is 10.

Inherited from superclass NB_GRAPH

- **legInterpreter** ↑

Sets the interpreter of the legend text. Must be a string.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **legLocation** ↑

Sets the location of the legend, must be string. The location can be:

> 'Best' : Same as 'NorthWest'.
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'below' : Below the plot (when many subplots, it will place the legend below all of them)
> 'middle' : Could only be use for 2 x 2 subplot, and then it places the legend between the upper and lower subplots.

Location that only works for the graph() method:

> 'outsideright' : Legend is placed outside the axes on the right side, and the axes is resized, so the axes and legend does not take up more space than the axes did in the first place. The legend is vertically centered.
> 'outsiderighttop' : Same as 'outsideright', but placed vertically top.

Inherited from superclass NB_GRAPH

- **legPosition** ↑

Sets the position of the legend. Must be a 1x2 double. Where the first element is the x-axis location and the second is the y-axis location. [xPosition yPosition]

Caution : Both elements must be between 0 and 1. And where we have that [0, 0] will be the bottom left location of the axes and [1, 1] is the top right position of the axes. (This position will be the top left position of the legend itself.)

Caution : This option overruns the legLocation property.

Inherited from superclass NB_GRAPH

- **legReorder** ↑

Set this property to reorder the legend. Either a string with {'default'} | 'inverse' or a double with how to reorder the legend. E.g. if you have 3 legends to plot the default index will be [1,2,3], the inverse ('inv') will be [3,2,1]. If you want to decide that the 3. legend should be first, then the 1. and 2., you can type in [3,1,2].

Caution : If a double is given it must have as many elements as there is plotted legends. I.e. if you give a empty string to one of the legends it will be excluded. Say you provide the property legends as {'s','t','f'} then the double must have size 1x3.

Inherited from superclass NB_GRAPH

- **legSpace** ↑

The vertical space between the texts of the legend. Must be a scalar. Default is 0.

Inherited from superclass NB_GRAPH

- **legendText** ↑

Sets the legend(s) of the given variable(s). Must be a cell array on the form: {'var1', 'Name', 'var2', 'Name2', ...}. Here we set the legends of the variables 'var1' and 'var2' to 'Name' and 'Name2' respectively. (The rest will be given the variable names as default).

To give a legend to a splitted line (second part) give; {...,'Var1(second)', 'LegendSplitted', ...} (Is case sensitiv)

Caution : legAuto should be set to 'on'. Which is not default. This is only important if the patch or fakeLegend property is used.

Caution : When given this property will overwrite the legends property

Caution : If the variable, scattergroup or candle is not found

```
no error will occure!
```

Caution : If you have used the patch or fakeLegend property these can be translated as well with this property

Inherited from superclass NB_GRAPH

- **legends** ↑

Give the legends of the plot. Must be given as a cell array of strings.

Default (it depends on the method you use) :

```
> 'graph'           : The variable names are used as the
                      default legends. (Given by the
                      variableToPlot and variableToPlotRight
                      properties)

> 'graphSubPlot'   : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)

> 'graphInfoStruct' : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)
```

Caution : The ordering of the legends is as follows:

- patch : See the patch property (only when legAuto is set to 'off')
- variablesToPlot : Then the all the variablesToPlot variables. When You have splitted lines, i.e. either using the dashedLine property or the lineStyles property, these can be set after all the variablesToPlot. Have in mind that the ordering is kept.
- variablesToPlotRight : Then the comes all the variables provided by the variablesToPlotRight. Then the splitted lines.
- fakeLegend : Then you can provide the legend text for the fake legend. (only when legAuto is set to 'off')

Example:

```
data      = nb_ts.rand('2012',10,2);
plotter = nb_graph_ts(data);
plotter.set('variablesToPlot',{'Var1'},...
            'variablesToPlotRight',{'Var2'},...
            'lineStyles',{'Var1',{'-'},{'2014','--'}},...
            'Var2',{'-'},{'2014','--'});
            'legends',{'Test','Test (--)','Test2','Test2 (--)'});
plotter.graph()
```

Inherited from superclass NB_GRAPH

- **lineStyles** ↑

Sets the line styles of the given variable(s). Must be a cell array on the form: {'var1', '--', 'var2', '---', ...}. Here we set the line styles of the variables 'var1' and 'var2' to '--' and '---' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Supported line styles are '--','--',':','.-','---

Caution : For nb_graph_ts and nb_graph_data it is possible to split the plotted line using the following syntax;
{'var1',{ '--', '2000Q1', '--' }} and {'var1',{ '--', 2, '--' }} respectively

Inherited from superclass NB_GRAPH

- **lineWidth** ↑

Sets the line width of the line plots. Must be a scalar.
Default is 2.5.

Inherited from superclass NB_GRAPH

- **lineWidths** ↑

Sets the line widths of the given variables. Must be a cell array on the form: {'var1', 1, 'var2', 1.5,...}. Here we set the line widths of the variables 'var1' and 'var2' to 1 and 1.5 respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Inherited from superclass NB_GRAPH

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_GRAPH

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_GRAPH

- **markerSize** ↑

Sets the size of the all markers. Must be a scalar. Default is 8.

Inherited from superclass NB_GRAPH

- **markers** [↑](#)

Sets the markers of the given variables. Must be a cell array on the form: {'var1', '^', 'var2', 'o', ...}. Here we set the markers of the variables 'var1' and 'var2' to '^' and 'o' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the table below on which markers that are supported.

'+'	: Plus sign
'o'	: Circle
'*'	: Asterisk
'.'	: Point
'x'	: Cross
's'	: Square
'd'	: Diamond
'^'	: Upward-pointing triangle
'v'	: Downward-pointing triangle
'>'	: Right-pointing triangle
'<'	: Left-pointing triangle
'p'	: Five-pointed star (pentagram)
'h'	: Six-pointed star (hexagram)

Inherited from superclass NB_GRAPH

- **noLabel** [↑](#)

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Both on the y-axis and x-axis. Default is 0. This is of course only have an effect if the nb_graph object have been given the xLabel or yLabel properties. (These are empty by default.)

Inherited from superclass NB_GRAPH

- **noLegend** [↑](#)

Set it to 1 if no legend is wanted, otherwise set it to 0. Default is 0.

Inherited from superclass NB_GRAPH

- **noTickMarkLabels** [↑](#)

Set to true (1) to remove tick marks and tick marks label of axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsLeft** [↑](#)

Set to true (1) to remove tick marks and tick marks label of the left side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsRight** ↑

Set to true (1) to remove tick marks and tick marks label of the right side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the graphs, otherwise set it to 0. Default is 0.

Set it to 2 if you want the dataNames{ii} as the title of the graph when using the graph method.

Inherited from superclass NB_GRAPH

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_GRAPH

- **numberOfGraphs** ↑

Number of graph produced by the graph methods. Not settable.

Inherited from superclass NB_GRAPH

- **page** ↑

Sets the page of the data object to plot. Only an option for graph() method. Default is [], i.e. plot all pages in separate figures.

Inherited from superclass NB_GRAPH

- **parameters** ↑

A struct with the parameters that can be used in evaluation of expressions in the graphSubPlots and graphInfoStruct methods.

Inherited from superclass NB_GRAPH

- **patch** ↑

Sets the patch property of the object. If set this will add a patch (color fill) between two variables.

Only one patch :

```
{'patchName','var1','var2',color}
```

More patches :

```
{'patchName1','var1Patch1','var2Patch1',color1 ,...  
'patchName2','var1Patch2','var2Patch2',color2,...}
```

Where the the color option(s) must either be a 1 x 3 double with the RGB colors or a string with the color name.

The first input will be the legend text of the patch. I.e. 'patchName', 'patchName1' and 'patchName2' will be the description text included in the legend. (If not the property legAuto is set to 'off'.)

Inherited from superclass NB_GRAPH

- **patchAlpha** ↑

Sets the transparency of the patches. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be used in combination with the saveName property.

Inherited from superclass NB_GRAPH

- **pieAxisVisible** ↑

Make the axes box visible ('on') or not ('off'), when doing pie or donut plot. 'on' is default.

- **pieEdgeColor** ↑

Sets the pie and donut edge color. Default is to not display edges (same as setting to 'none'). Can be a one line char with the colors that can be interpreted by nb_plotHandle.interpretColor, or a double with size 1x3 with the RGB colors.

- **pieExplode** ↑

Sets the element(s) of the pie or donut chart which should explode. Must be a cellstr of the variables to explode. E.g. {'Var1','Var2',...}

- **pieLabelsExtension** ↑

Extension of the pie or donut labels, e.g. 'mrd.kr' or '%'. Must be a string.

- **pieOrigoPosition** ↑

Set the origo position of pie or donut plot. Default is [-0.5,0]. Must either be empty or a 1x2 double.

- **pieTextExplode** ↑

Set this property to make text labels move more from the center of the pie or donut chart. Must be a cellstr of the variables to explode the text of. E.g. {'Var1','Var2',...}

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_GRAPH

- **plotType** ↑

Sets the plot type. Either:

- > 'line' : If line plot(s) is wanted. Default.
- > 'stacked' : If stacked bar plot(s) is wanted.
- > 'grouped' : If grouped bar plot(s) is wanted.
- > 'dec' : If decomposition plot(s) is wanted. Which is a bar plot with also a line with the sum of the stacked bars.
- > 'area' : If area plot(s) is wanted.
- > 'radar' : If radar plot(s) is wanted.
- > 'candle' : If candle plot(s) is wanted.
- > 'scatter' : If scatter plot(s) is wanted.
- > 'pie' : If pie plot(s) is wanted.
- > 'donut' : If donut plot(s) is wanted.
- > 'image' : If you want to plot the data as an image mapped to the colors found by the property colorMap.

- **plotTypes** ↑

Sets the plot type of the given variables. Must be a cell array on the form: {'var1', 'line', 'var2', 'grouped', ...}. Here we set the plot type of the variables 'var1' and 'var2' to 'line' and 'grouped' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Only the plot types 'line', 'stacked', 'grouped', 'dec' and 'area' are supported.

- **position** ↑

Sets the position of the axes, must be an 1x4 double. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

For the graph method graphSubPlots the position input can be given a $1 \times N$ cell array, where each element is a 1×4 double.
 $N = \text{obj}.\text{subPlotSize}(1) * \text{obj}.\text{subPlotSize}(2)$.

Inherited from superclass NB_GRAPH

- **radarFontColor** ↑

The color of the font of the labels. Either a 1×3 double with the color of all the labels or a $\text{size}(xData, 1) \times 3$ with the color of each label (RGB colors). Can also be a string with the color names to use on all labels or a cellstr with the color names of each label. Size must be $1 \times \text{size}(xData, 1)$

- **radarNumberOfIsoLines** ↑

The number of isocurves of the radar plot. Must be an integer. Default is 10.

- **radarRotate** ↑

Rotate the radar. In radians. Must be a scalar. Default is 0.

- **radarScale** ↑

Sets the axes limits, so the radar plot is scaled properly. (Also set the size of the plotted radar). Must be a 1×2 double. Default is [2, 1.25].

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

Inherited from superclass NB_GRAPH

- **scatterLineStyle** ↑

Sets the scatter line style. Must be string. See 'lineStyles' for supported inputs. (Sets the line style of all scatter plots)

- **scatterTypes** ↑

The types used for the scatter plot. Must be a 1×2 cellstr array. The first element will be the type plotted against the x-axis, while the second element will be the type plotted against the left y-axis. E.g:

```
{'type1','type2'}
```

- **scatterTypesRight** ↑

The types used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the type plotted against the x-axis, while the second element will be the type plotted against the right y-axis. E.g:

```
{'type1','type2'}
```

- **scatterVariables** ↑

Sets the variables to be used for the scatter plot plotted against the left axes. Must either be a nested cell array. E.g:

```
{'scatterGroup1',{'var1','var2','...'},...
'scatterGroup2',{'var10','var11','...'})
```

Be aware : Each variable will result in one point in the scatter.

- **scatterVariablesRight** ↑

Sets the variables to be used for the scatter plot plotted against the right axes. Must either be a nested cell array. E.g:

```
{'scatterGroup1',{'var1','var2','...'},...
'scatterGroup2',{'var10','var11','...'})
```

Be aware : Each variable will result in one point in the scatter.

- **shading** ↑

The shading option of the axes background:

- 'grey' : Shaded grey background
- 'none' : Background color given by the color property.
- A n x m x 3 double with the background color. n is the number of horizontal pixels, m is the number of vertical pixels and 3 means the RGB colors.

Inherited from superclass NB_GRAPH

- **subPlotSize** ↑

Sets the number of subplots per figure. Must be a 1 x 2 double with the number of subplot rows as the first element and the number of subplot columns as the second element. Only an option of the graphSubPlots() and graphInfoStruct() methods.

Default is [2, 2].

Inherited from superclass NB_GRAPH

- **subPlotSpecial** ↑

Set it to 1 if you want the 1x2 and 2x1 subplots to better fit for usage in presentations. Default is 0, i.e. use MATLAB default positions.

When set to 1 this class uses the nb_subplotSpecial instead of nb_subplot to find the positions of the subplots.

Inherited from superclass NB_GRAPH

- **sumTo** ↑

Sets a number which the area or the stacked bar plot should sum to. Must be a scalar. I.e. if 100 is given. The sizes of the bars will be the percentage share of the total sum.

Inherited from superclass NB_GRAPH

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_GRAPH

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_GRAPH

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_GRAPH

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **typesToPlot** ↑

Sets the types to plot for each plotted variables. Must be a cellstr. E.g. {'type1','type2',...}

If the plotType property is set to 'pie', selecting more plot types will create more graphs (one per type).

- **typesToPlotRight** ↑

Sets the types to plot for each plotted variables on the right axes. Must be a cellstr. E.g. {'type1','type2',...}. Only an option if plotType is set to 'grouped', and barOrientation is set to horizontal, and some variables are given to the variablesToPlotRight property.

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_GRAPH

- **variablesToPlot** ↑

Sets the variables to plot against the left (or both) axes. Must be a cell array of strings with the variable names. I.e. {'var1', 'var2',...}.

Not an option for the graphInfoStruct(...), which use the property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **variablesToPlotRight** ↑

Sets the variables to plot against the right axes. Must be a cell array of strings with the variable names. I.e. {'var1', 'var2',...}.

Not an option for the graphInfoStruct(...), which use the property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **verticalLine** ↑

Sets the type(s) of where to place a vertical line(s). Must be a cellstr array with all the types which should have a vertical line, either 'type' or {'type1','type2',...}.

Each element of the cell could also be given as a cell with two types, i.e. {'type1','type2'}. Then the vertical line

will be placed between the given types.

If plotType is set to 'scatter' scalars can be used instead.

- **verticalLineColor** ↑

Sets the color(s) of the given vertical line(s). Must either be an M x 3 double with the RGB color(s) or a 1 x M cell array of strings with the color name(s). Where M must be less than the number of plotted vertical lines.

If less or no color(s) is/are set by this property the default color(s) for the rest or all the vertical line(s) is/are [0 0 0]. I.e. MATLAB black.

- **verticalLineLimit** ↑

Sets the y-axis limit(s) of the given vertical line(s). Must either be a 1 x M cell array of 1 x 2 double with the upper and lower y-axis limit(s). Where M must be less than the number of plotted vertical lines.

If less or no limit(s) is/are set by this property the default limit for the rest or all the vertical line(s) is/are the full height of the graph.

- **verticalLineStyle** ↑

Sets the line style(s) of the given vertical line(s). Must either be a 1 x M cell array of strings with the style(s). Where M must be less than the number of plotted vertical lines.

If less or no style(s) is/are set by this property the default line style is/are used for the rest or all the vertical line(s) is/are '--'.

- **verticalLineWidth** ↑

Sets the line width of (all) the vertical line(s). Must be a scalar. Default is 1.5.

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_GRAPH

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel
property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' |
'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **xLim** ↑

Sets the x-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Both or one of them can be
set to nan, i.e. the default limits will be used) E.g. [0 6],
[nan, 6] or [0, nan].

Only an option when the plotType property is set to 'scatter'.

- **xScale** ↑

Sets the scale of the x-axis. Either {'linear'} | 'log'.

Only an option when plotType is set to 'scatter'.

- **xSpacing** ↑

Sets the x-ticks spacing. Must be a scalar.

- **xTickLabelAlignment** ↑

The alignment of the x-axis tick marks labels. {'below'} |
'middle'.

- **xTickLabelLocation** ↑

The location of the x-axis tick marks labels. Either {'bottom'} | 'top' | 'baseline'. 'baseline' will only work in combination with the xtickLocation property set to a double with the basevalue.

- **xTickLabels** ↑

Sets the x-axis tick mark labels. To translate the default tick mark label 'Var1' and 'Var2' you can use;

```
{'Var1','Label1','Var2','Label2'}
```

If given as a empty cell, default x-axis tick marks will be used, which is default.

This property can be used to create multi-lined x-axis tick mark labels. To do this give a multi-row char to the labels you want to make multi-lined.

Inherited from superclass NB_GRAPH

- **xTickLocation** ↑

The location of the x-axis tick marks. Either {'bottom'} | 'top' | double. If given as a double the tick marks will be placed at this value (where the number represent the y-axis value).

- **xTickRotation** ↑

Sets the rotation of the x-axis tick mark labels. Must be a scalar with the number of degrees the labels should be rotated. Positive angles cause counterclockwise rotation.

- **yDir** ↑

Sets the direction of the y-axis of the plot. Must be a string. Either 'normal' or 'reverse'. Default is 'normal'.

Caution: If the variablesToPlotRight is not empty this setting only sets the left y-axis direction.

- **yDirRight** ↑

Sets the direction of the right y-axis of the plot. Either 'normal' or 'reverse'. Default is 'normal'. (Only when the variablesToPlotRight property is not empty.)

- **yLabel** ↑

Sets (add) the text of the y-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **yLabelFontSize** ↑

Sets the font size of the y-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_GRAPH

- **yLabelFontWeight** ↑

Sets the font weight of the y-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **yLabelInterpreter** ↑

The interpreter used for the given string given by the xLabel
property.

```
> 'latex' : For mathematical expression
> 'tex'   : Latex text interpreter,
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **yLabelRight** ↑

Sets the text of the y-axis label. (Only on the right side.)
Must be a string. Takes the same font option as the yLabel
property

Inherited from superclass NB_GRAPH

- **yLim** ↑

Sets the y-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Both or one of them can be
set to nan, i.e. the default limits will be used) E.g. [0 6],
[nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty
this setting only sets the left y-axis limits.

Only an option for the graph(...) and graphSubPlots() methods.
(For the graphInfoStruct(...) method use the property
GraphStruct.)

- **yLimRight** ↑

Sets the right y-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Only when the
variablesToPlotRight property is not empty.) (Both or one of
them can be set to nan, i.e. the default limits will be used)
E.g. [0 6], [nan, 6] or [0, nan].

Only an option for the graph(...) and graphSubPlots() methods.
(For the graphInfoStruct(...) method use the property
GraphStruct.)

- **yOffset** ↑

The y-axis tick mark labels offset from the axes

- **yScale** ↑

Sets the scale of the y-axis. {'linear'} | 'log'. (Both the left and the right axes if nothing is plotted on the right axes.)

- **yScaleRight** ↑

Sets the scale of the right y-axis. {'linear'} | 'log'. (Has only something to say if something is plotted on the right axes)

- **ySpacing** ↑

Sets the spacing between y-axis tick mark labels of the y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow the frequency of the xTickFrequency property if setted, else it will follow the frequency of the data provided. (Except when dealing with daily data, because then the default xTickFrequency is 'yearly')

Caution: If the variablesToPlotRight property is not empty this property only sets the spacing between ticks of the left y-axis.

- **ySpacingRight** ↑

Sets the spacing between y-axis tick mark labels of the right y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow

Methods:

- [createFigure](#)
- [get](#)
- [getCSOptions](#)
- [getCode](#)
- [getData](#)
- [getGenericOptions](#)
- [getPlottedTypes](#)
- [getPlottedVariables](#)
- [graph](#)
- [graphInfoStruct](#)
- [graphSubPlots](#)
- [isempty](#)
- [merge](#)
- [resetDataSource](#)
- [saveData](#)
- [saveFigure](#)
- [set](#)
- [struct](#)
- [update](#)

► **createFigure** ↑

```
createFigure(obj)
```

Description:

Create figure to plot in.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► **get ↑**

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_graph_cs

Input:

- obj : An object of class nb_graph_cs
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_graph_cs object. Be aware that all the the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_graph_cs object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth S. Paulsen

► **getCSOptions ↑**

```
options = getCSOptions(obj)
```

Description:

A static method of nb_graph_cs.

Container with the properties specific to nb_graph_cs

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and function handles with the set function in the fourth.

Examples:

```
options = getCSOptions(obj);
```

See also:

[set](#), [getGenericOptions](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

► getCode ↑

```
[code,numOfGraphs] = getCode(obj,frameTitle,frameSubTitle,...  
footer,source,pageNumber,theme)
```

Description:

Get the latex code when including a graph in a slide object produced by an object of class nb_graph_ts.

This method is called from the nb_Slide class, when making a beamer slide in latex.

Input:

- obj : An object of class nb_graph_ts
- frameTitle : A string with the frame title of the slide.
- frameSubTitle : A string with the frame subtitle of the slide.
- footer : A cellstr with the footers of the slide
- source : A cellstr with the sources of the slide
- pageNumber : Give 1 if the page number should be included.
- theme : The beamer theme used. Default is '' (use default beamer theme).

Output:

- code : The latex code as a char.
- numOfGraphs : The number of graphs produced by the input object of class nb_graph_ts. As an integer.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► getData ↑

```
data = getData(obj)
```

Description:

Get the data of the graph

Input:

- obj : An object of class nb_graph_cs

Output:

- data : As an nb_cs object

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

► getGenericOptions ↑

```
options = getGenericOptions(obj)
```

Description:

A static method of nb_graph.

Container with the intersect of options for subclasses of nb_graph.

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getGenericOptions(obj);
```

See also:

[set](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

Inherited from superclass `NB_GRAPH`

► **getPlottedTypes** ↑

```
types = getPlottedTypes(obj)
```

Description:

Get the plotted types of the graph

Input:

- obj : An object of class `nb_graph_cs`

Output:

- types : As a cellstr

Example:

```
types = obj.getPlottedTypes();
```

Written by Kenneth S. Paulsen

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the plotted variables of the graph

Input:

- obj : An object of class nb_graph_cs
- forLegend : true or false (default)

When forLegend is used the variables are listed as needed for the legend, but without fake legends and patches, and duplicates are not removes.

Output:

- vars : A cellstr

Example:

```
vars = obj.getPlottedVariables();
```

Written by Kenneth S. Paulsen

► **graph ↑**

```
graph(obj)
```

Description:

Create a graph or more graphs

Graphs all the variables given by the variablesToPlot and the variablesToPlotRight properties in one plot, and does that for all the datasets of the DB property. I.e. one new plot for each dataset (page) of the DB property of the nb_graph_cs object.

Input:

- obj : An object of class nb_graph_cs

Output:

The graph plotted on the screen.

Examples:

```
data = nb_cs([2;1;2],'',{'type1','type2','type3'},'Var1');
obj = nb_graph_cs(data);
obj.graph();
```

See also:

[nb_graph_cs.set](#)

Written by Kenneth S. Paulsen

► graphInfoStruct ↑

```
graphInfoStruct(obj)
```

Description:

Create graphs based on the property GraphStruct. If it is empty nothing wil happen.

This method makes it easy to set up graph packages. For example graphing output from models.

See the documentation NB toolbox for more on this method.

Input:

- obj : An object of class nb_graph_cs, where the property GraphStruct contains the information on how to plot the objects data.

Output:

The wanted graphs plotted on the screen.

Examples:

```
obj.graphInfoStruct();
```

Written by Kenneth S. Paulsen

► graphSubPlots ↑

```
graphSubPlots(obj)
```

Description:

Create graphs of all the variables given in variablesToPlot property, in different subplots. (Each dataset against each other). The number of subplot is given by the property subPlotSize.

Input:

- obj : An object of class nb_graph_cs

Output:

The wanted graphs plotted on the screen.

Examples:

```
data = nb_cs([2 3 2;1 4 5;2 3 2],'',{'type1','type2','type3'},...  
            {'Var1','Var2','Var3'});  
obj = nb_graph_cs(data);  
obj.graphSubPlots();
```

Written by Kenneth S. Paulsen

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_graph_cs object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_graph_cs

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **merge** ↑

```
out = merge(obj,varargin)
```

Description:

Default: This tries to merge the data of plotter objects that uses the graphSubPlots, or the graphInfoStruct.

Caution: All objects must share the exact same variables!

'graph' is given: In this case it will try to merge the data of all nb_graph objects, and concatenate the variableToPlot property. If the property variableToPlot is empty for all objects, it will also be returned as empty, i.e. plot all series in the data.

Caution : Merging of any of the other properties then DB and variableToPlot is not done!

Input:

- obj : An object of class nb_graph, which includes the classes nb_graph_ts, nb_graph_cs and nb_graph_data.

Optional input:

- 'graph' : Give this input to merge the graphs in the way that is described in the description of this method.
- varargin : Each input must be an object of class nb_graph.

Output:

- out : An object of class nb_graph.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► resetDataSource ↑

```
[message,err] = resetDataSource(obj,varargin)
```

Description:

Reset the data source of the nb_graph_cs object.

Input:

- obj : An object of class nb_graph_cs
- dataSource : An object of class nb_cs.
- updateProps : If the variablesToPlot should be updated to be equal to dataSource.variables + typesToPlot should be updated to be equal to dataSource.types.

Can also be 'gui'. Then all properties will be checked against the new dataset, and if the new dataset does not comply with the properties, the properties will be updated!

Written by Kenneth S. Paulsen

► saveData ↑

```
obj = saveData(obj,filename)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_graph_cs
- filename : A string with the saved output name

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

► **saveFigure ↑**

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_graph
- extraName : If you want to add some extra name to the saveName property of the nb_graph_data object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Examples:

```
data = nb_data([2;1;2],'',1,'Var1');
obj = nb_graph_data(data);
obj.graph();
saveFigure(obj,'test','pdf');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **set** ↑

```
set(obj,varargin)
```

Description:

Set properties of the nb_graph class and its subclasses.

See documentation NB Toolbox or type `help('nb_graph')` for more on the properties of the class.

Input:

- `obj` : An object that is a subclass of nb_graph
- `varargin` : `'propertyName'`,`propertyValue`,...

Output:

No actual output, but the input objects properties will have been set to their new value.

Examples:

```
data = nb_ts([2;1;2],'', '2012Q1', 'Var1');
obj = nb_graph_ts(data);
obj.set('startGraph','2012Q1','endGraph','2012Q3');
obj.set('crop',1);
```

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct

Input:

- `obj` : An object of class nb_graph_cs

Output:

- `s` : A struct

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the figure

See the update method of the nb_cs class for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_graph_cs

Output:

- obj : An object of class nb_graph_cs, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ **nb_graph_data**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_data(data)
```

Superclasses:

handle, nb_graph

Description:

This is a class for making dataseries graphics.

Constructor:

```
obj = nb_graph_data(data)
```

Input:

- data :

The input must be one of the following:

> An object of class nb_data. E.g. nb_graph_data(data)

> A excel spreadsheet which could be read by the nb_data
class. E.g. nb_graph_data('excelName')

Output:

- obj : An object of class nb_graph_data

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#), [nb_graph_adv](#), [nb_graph_subplot](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- DB
- a4Portrait
- addSpace
- annotation
- areaAccumulate
- axesFast
- axesFontSizeX
- axesLineWidth
- axesScaleFactor
- barAlpha1
- barBlend
- barPeriods
- barShadingDirection
- barWidth
- baseLineColor
- baseLineWidth
- candleIndicatorColor
- candleMarker
- candleWidth
- colorBarAtEnd
- colorOrder
- colors
- dashedLine
- endBarWidth
- excelFooterEng
- excelTitleEng
- factor
- GraphStruct
- addAdvanced
- alignAxes
- areaAbrupt
- areaAlpha
- axesFontSize
- axesFontWeight
- axesPrecision
- axesScaleLineWidth
- barAlpha2
- barLineWidth
- barShadingColor
- barShadingObs
- baseLine
- baseLineStyle
- baseValue
- candleIndicatorLineStyle
- candleVariables
- code
- colorMap
- colorOrderRight
- crop
- displayValue
- endGraph
- excelFooterNor
- excelTitleNor
- fakeLegend

- fanColor
- fanLegend
- fanLegendLocation
- fanMethod
- fanVariable
- figureName
- figureTitle
- findAxisLimitMethod
- fontName
- fontUnits
- grid
- highlight
- horizontalLine
- horizontalLineStyle
- language
- legBox
- legColumnWidth
- legFontColor
- legInterpreter
- legPosition
- legSpace
- legends
- lineStyles
- lineWidths
- lookUpMatrix
- markerSize
- missingValues
- fanDatasets
- fanLegendFontSize
- fanLegendText
- fanPercentiles
- figureColor
- figurePosition
- fileFormat
- flip
- fontScale
- graphStyle
- gridLineStyle
- highlightColor
- horizontalLineColor
- horizontalLineWidth
- legAuto
- legColor
- legColumns
- legFontSize
- legLocation
- legReorder
- legendText
- lineStop
- lineWidth
- localVariables
- mYLim
- markers
- nanAfterObs

- nanBeforeObs
- noLabel
- noTickMarkLabels
- noTickMarkLabelsRight
- normalized
- obs
- parameters
- patchAlpha
- plotAspectRatio
- plotTypes
- saveName
- scatterObs
- scatterVariables
- shading
- startGraph
- subPlotSpecial
- tag
- titleAlignment
- titleFontWeight
- titlePlacement
- variableToPlotX
- variablesToPlotRight
- verticalLineColor
- verticalLineStyle
- xLabel
- xLabelFontSize
- xLabelInterpreter
- nanVariables
- noLegend
- noTickMarkLabelsLeft
- noTitle
- numberOfGraphs
- page
- patch
- pdfBook
- plotType
- position
- scatterLineStyle
- scatterObsRight
- scatterVariablesRight
- spacing
- subPlotSize
- sumTo
- title
- titleFontSize
- titleInterpreter
- userData
- variablesToPlot
- verticalLine
- verticalLineLimit
- verticalLineWidth
- xLabelAlignment
- xLabelFontWeight
- xLabelPlacement

- `xLim`
- `xTickLabelAlignment`
- `xTickLabels`
- `xTickRotation`
- `yDir`
- `yLabel`
- `yLabelFontSize`
- `yLabelFontWeight`
- `yLabelRight`
- `yLimRight`
- `yScale`
- `ySpacing`
- `xScale`
- `xTickLabelLocation`
- `xTickLocation`
- `xTickStart`
- `yDirRight`
- `yLabelInterpreter`
- `yLim`
- `yOffset`
- `yScaleRight`
- `ySpacingRight`

- **`DB`** 

All the data given to the `nb_graph_data` object collected in a `nb_data` object. Should not be set using `obj.DB = data!` Instead see the `nb_graph_data.resetDataSource` method.

- **`GraphStruct`** 

Sets the information on which variables and how to plot them when using the `graphInfoStruct(...)` method of this class.

Must be an .m-file with the code on how the structure of graphing information should be initialized or as a structure with the graphing information.

Only the `graphInfoStruct(...)` method uses this property. See the documentation of the NB toolbox for more on this input.

Inherited from superclass `NB_GRAPH`

- **`a4Portrait`** 

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass `NB_GRAPH`

- **`addAdvanced`** 

Set to false to prevent adding advanced components.

Inherited from superclass `NB_GRAPH`

- **`addSpace`** 

Sets the space before and/or after in the x direction of the graph [`addedSpaceBefore` `addedSpaceAfter`]. E.g. add one period before and after [1,1].

Inherited from superclass `NB_GRAPH`

- **alignAxes** ↑

Align axes at a given base value. Must be set to a scalar double.
E.g. 0.

Inherited from superclass NB_GRAPH

- **annotation** ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the nb_annotation class for more. If you give more objects they must be collected in a cell array.

Examples can be found here:

\NBTOOLBOX\Examples\Graphics\nb_annotationExamples.m

Inherited from superclass NB_GRAPH

- **areaAbrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan. Default is false.

Inherited from superclass NB_GRAPH

- **areaAccumulate** ↑

Set if the areas should be accumulated or not. Default is true.

Inherited from superclass NB_GRAPH

- **areaAlpha** ↑

Sets the transparency of the area chart. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **axesFast** ↑

Set the if you want graphs to be produced fast with a loss of functionality and look or not. When empty default for the graph method is false, while for the graphSubPlots and graphInfoStruct methods are true.

Inherited from superclass NB_GRAPH

- **axesFontSize** ↑

Sets the font size of the axes tick mark labels, default is 12. Must be a scalar.

Inherited from superclass NB_GRAPH

- **axesFontSizeX** ↑

Sets the font size of the x-axis tick mark labels, default is []. I.e. use the axesFontSize property instead. Must be a scalar.

This property will not be scaled when fontUnits are changed, as is the case for axesFontSize.

Caution: Will not set the x-axis font size when plotType is set to scatter.

Inherited from superclass NB_GRAPH

- **axesFontWeight** ↑

Sets the font weight of the axes tick mark labels. Must be a string. Default is 'normal'. Either 'normal', 'bold', 'light' or 'demi'

Inherited from superclass NB_GRAPH

- **axesLineWidth** ↑

Sets the line width of the axes, default is 0.5. Must be a set to a scalar double greater than 0.

Inherited from superclass NB_GRAPH

- **axesPrecision** ↑

Sets the precision/format of the rounding of number on the axes.

See the precision input to the nb_num2str function. Default is [], i.e. to call num2str without additional inputs.

Inherited from superclass NB_GRAPH

- **axesScaleFactor** ↑

Set the scaling factor used when axesScaleLineWidth is set to true.

Inherited from superclass NB_GRAPH

- **axesScaleLineWidth** ↑

Scale line width to axes height. true or false (default). If axesScaleFactor is set to a scalar number this will be the scaling factor used instead of the automatic scaling factor.

Inherited from superclass NB_GRAPH

- **barAlpha1** ↑

Set the alpha blending parameter 1, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barAlpha2** ↑

Set the alpha blending parameter 2, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barBlend** ↑

Set to true to do alpha blending instead of shading if shading options is used. Default is false.

Inherited from superclass NB_GRAPH

- **barLineWidth** ↑

Set the bar line width without affecting any other line widths. Default is empty, i.e. use the lineWidth property.

Inherited from superclass NB_GRAPH

- **barPeriods** ↑

The periods for where to plot the bar. E.g 3 or [3,4]. Only an option for the graphSubPlots() method. The lineStop must be set to make this option work. Default is [].

- **barShadingColor** ↑

The color shaded bars are interpolated with. Default is 'white'. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **barShadingDirection** ↑

Sets the angle the shading should be vertical of. Either {'north'} | 'south' | 'west' | 'east'. As a string. Default is 'north'.

Inherited from superclass NB_GRAPH

- **barShadingObs** ↑

Sets the date from which the bar plot should be shaded. As an integer. Default is [].

- **barWidth** ↑

Sets the width of the bar plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **baseLine** ↑

If 0 is given the base line will not be plotted, otherwise it will be plotted.

Inherited from superclass NB_GRAPH

- **baseLineColor** ↑

Sets the color of the base line. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **baseLineStyle** ↑

Sets the line style of the base line. As a string.
Either {'-' | '--' | '---' | ':' | '-.' | 'none'.

Inherited from superclass NB_GRAPH

- **baseLineWidth** ↑

Sets the width of the base line. Must be a scalar.

Inherited from superclass NB_GRAPH

- **baseValue** ↑

Sets the base value used for bar and area plot. (Sest also the
y-axis base value for the base line)

Inherited from superclass NB_GRAPH

- **candleIndicatorColor** ↑

Sets the color of the indicator of the candle. Must either be a
1 x 3 double with the RGB colors or a string with the color
name.

Inherited from superclass NB_GRAPH

- **candleIndicatorLineStyle** ↑

Sets the line style of the indicator of the candle. Must be a
string. Default is '-'.

Inherited from superclass NB_GRAPH

- **candleMarker** ↑

Sets the marker of the indicator of the candle. Must be a
string. Default is 'none'.

Inherited from superclass NB_GRAPH

- **candleVariables** ↑

Sets which variables are going to be plotted as open, close,
high, low and indicator. Must be a cellstr. E.g.

{'open','var1','close','var2',...}

If a type (e.g. 'low') is not given it will not be plotted.

Meaning of each type:

> 'close' : The lowest values of the plotted patches of
the candle.

> 'high' : The highest values of the plotted candles.

> 'indicator' : The value for where to plot a horizontal line
(across the patch). E.g. the mean value.

> 'open' : The highest values of the plotted patches of the candle.

> 'low' : The lowest values of the plotted candles.

Inherited from superclass NB_GRAPH

- **candleWidth** [↑](#)

Sets the width of the candle plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **code** [↑](#)

If you set this to a file name, this file will be evaluated after all the plotting is done. Must be a string.

This could be nice for adding special annotation, lines and so on. In this file you can use any MATLAB function, but some strange error message can occur if you use local variables which is already defined (So use long and descriptive variable names in the provided code to prevent this).

This property is only an option for the method graph.

Nice to know:

> The current nb_figure object can be reached with
get(obj,'figureHandle')

> The current nb_axes object can be reached with
get(obj,'axesHandle')

Caution : Be aware that the name of the file must be unique and cannot be a MATLAB function or a local variable.

Caution : The file must be a MATLAB .m file

Inherited from superclass NB_GRAPH

- **colorBarAtEnd** [↑](#)

A string with the color specification to use for the bars when the line with bars at the end plot type is triggered. One of 'nb' | 'red' | 'green' | 'yellow' | ''. Default is 'nb'. Use the lineStop and barPeriods properties to trigger this plot type.

- **colorMap** [↑](#)

Sets the colormap used when plotType is set to 'image' and fanMethod is set to 'graded'. Must be given as n x 3 double or the path to a .mat file that contain the colormap. Default is given by nb_axes.defaultColorMap.

For an example of a supported MAT file see:

- ...\\Examples\\Graphics\\colorMapNB.mat

Inherited from superclass NB_GRAPH

- **colorOrder** [↑](#)

Sets the color order of the plotted data (Left axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the left axes.

Inherited from superclass NB_GRAPH

- **colorOrderRight** [↑](#)

Sets the color order of the plotted data (Right axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the right axes.

Inherited from superclass NB_GRAPH

- **colors** [↑](#)

Sets the colors of the given variable(s). Must be a cell array on the form: {'var1', 'black', 'var2', [0.2 0.2 0.2],...}. Here we set the colors of the variables 'var1' and 'var2' to 'black' and [0.2 0.2 0.2] (RGB color) respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Caution: For the graphInfoStruct() method you need to select the variables from obj.DB.dataNames instead!

Inherited from superclass NB_GRAPH

- **crop** [↑](#)

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_GRAPH

- **dashedLine** [↑](#)

Sets the date at which all the plotted lines will be dashed. Must be given as an integer with the obs (or a double when the variableToPlotX is given)

- **displayValue** [↑](#)

Set to false to not display observation values when holding the mouse over the graph. Defualt is true.

Inherited from superclass NB_GRAPH

- **endBarWidth** [↑](#)

The width of the bars when the line with bars at the end plot type is triggered. Use the lineStop and barPeriods properties to trigger this plot type. Default is 3.

- **endGraph** ↑

Sets the end obs of the graph. Either as an integer. If the given obs is after the end obs of the data, the data will be appended with nan values. I.e. the graph will be blank for these obs.

Not used when variableToPlotX property is set, use xLim instead.

- **excelFooterEng** ↑

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. English version.

Inherited from superclass NB_GRAPH

- **excelFooterNor** ↑

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. Norwegian version.

Inherited from superclass NB_GRAPH

- **excelTitleEng** ↑

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). English version.

Inherited from superclass NB_GRAPH

- **excelTitleNor** ↑

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). Norwegian version.

Inherited from superclass NB_GRAPH

- **factor** ↑

Set this property to multiply the data with a given factor.
Must be a scalar.

Inherited from superclass NB_GRAPH

- **fakeLegend** ↑

Set this property if you want to add a legend of none plotted variable. You can set this property. Must be a cell on the form: {'legendName', settings,...}, where the legendName will be text of the fake legend. And the settings input must also be a cell array with the optional inputs {...,'propertyName', propertyName,...}. These settings will set the appearance of the fake legend. The following propertyNames are supported:

```
> 'type'      : Either 'line' or 'patch'.

> 'color'     : Sets the color of the fake legend. Default is
                 'black'. Must either be a 1 x 3 double with
                 the RGB colors or a string with the color
                 name. If shaded patch is wanted it must be
                 2 x 3 double with the RGB colors or a 1 x 2
                 cellstr with the color names to use.

> 'direction' : Sets the direction of the shading. Either
                 {'north'} | 'south' | 'west' | 'east'. As a
                 string. Default is 'north'. Only an option
                 when 'type' is set to 'patch'.

> 'edgeColor'  : Sets the color of the edge of the patch.
                 Default is the same as the 'color' option.
                 Only an option when 'type' is set to 'patch'.

> 'lineStyle'   : Sets the line style of the fake legend. As a
                 string. Either {'-' } | '--' | '---' | ':' |
                 '-.' | 'none'. Default is a normal line.

                 Will set the edge line style if the 'type'
                 is set to 'patch'.

> 'lineWidth'   : Sets the line width of the fake legend.
                 Default is 2.5. Must be a scalar.

> 'marker'     : Sets the marker of the fake legend. Default is
                 'none'. Only an option when 'type' is set to
                 'line'. Lookup LineSpec in the MATLAB help
                 meny for more on the supported marker types.
```

Caution : Even if you don't want to set any of the optional input. The settings input must be given. I.e an empty cell, i.e. {}.

Inherited from superclass NB_GRAPH

- **fanColor** ↑

Sets the colors of the fan charts. Must be a matrix of size; number of fan layer x 3, with the RGB color specifications.

or

A string with the basis color for the fan colors. Either 'red', 'blue', 'green', 'yellow', 'magenta', 'cyan' or 'white'.

Be aware that the classification of the color of the given

"basecolor" is based on the property fanPercentiles, so if you have more or less than 4 layers you must change this property to have the same size as the number of layers. (This is the case even if you don't use it to calculate the percentiles of a given datasets. Given by the property fanDatasets)

Default is the nb colors.

Caution: If fanMethod is set to 'graded' the color used are those assign to the colorMap property.

- **fanDatasets** ↑

Sets the data of the added fan layers.

If you have the data of the different layer in separate datasets. Then the input to this property must be a 2 * number of layers cell matrix. Can be excel spreadsheets or .mat files.

or

You can also give the data with all the simulated data or fan layers in a nb_cs object. (Each page one simulation or each page one layer).

Remember that the percentiles for the fan charts is set by the property fanPercentiles, default is [.3, .5, .7, .9].

The fan layers are added differently given the method you use.

> graph(...): Adds the fan given in this property to the last variable given in the property variablesToPlot.
(If not the fanVariable property is set.) If you graph more datasets (in separate figures), the fan layers will be the same for all of them.
Set the fanVariables property if that is not wanted.

> graphSubPlots(...): Adds the fan given in this property to the corresponding variable in the last given datasets (last page of the nb_cs object given by the property DB) of each subplot.

-graphInfoStruct(...): Adds the fan given by this property to each variable of each subplot. Added for the last given dataset. (Last page of the nb_cs object given by the property DB.)

- **fanLegend** ↑

Set this property if you want to add a MPR styled legend of the fan layers. Set it to 1 if wanted. Default is not (0).

- **fanLegendFontSize** ↑

Sets the font size of the fan legend. Default is 12. Must be a scalar.

- **fanLegendLocation** ↑

Sets the location of the fan legend. Locations are:

```
> 'Best', (same as 'North'.)
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'outside' : Places the legend outside the axes, one the
               right side. (If you create subplots, this is
               the only option which will look good.)
```

You can also give a 1×2 double vector with the location of where to place the legend. First column is the x-axis location and the second column is the y-axis location. Both must be between 0 and 1.

- **fanLegendText** ↑

Set this property if you want to give the text over the colored boxes of the MPR looking fan legend, you must give them as a cell. (Must have same size as number of layers.) Default is the percentiles with the % sign added. E.g.
{'30%', '50%', '70%', '90%'}.

- **fanMethod** ↑

{'percentiles' | 'hdi' | 'graded'}; Which type of fan chart is going to be made. 'percentiles' are produced by using percentiles, 'hdi' use highest density intervals, while 'graded' creates a graded fan chart of the colors specified with the colorMap property.

- **fanPercentiles** ↑

Sets the percentiles you want to calculate (if the fanDatasets property is representing simulation) or specify the percentiles you have given (if the fanDatasets property is representing already calculated percentiles). Must be given as a double vector. Default is [0.3 0.5 0.7 0.9].

This property is also the default base for the text of the fan legend. (Where the percentiles are given in percent.)

- **fanVariable** ↑

Sets the variable you want to add the fan layers to. Must be given as a string with the variable name. Default is to use the last variable in the variablesToPlot property.

- **figureColor** ↑

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_GRAPH

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_GRAPH

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_GRAPH

- **figureTitle** ↑

Sets if figure titles is wanted on the figures when using the method graphInfoStruct. The fieldnames of the GraphStruct property will be used as figure title. Default is 0 (not include).

If set to a one line char this property can be used for the plotting method graphSubPlots.

Inherited from superclass NB_GRAPH

- **fileFormat** ↑

Sets the save file format. Either 'eps', 'jpg', 'pdf' or 'png'. 'pdf' is default.

Inherited from superclass NB_GRAPH

- **findAxisLimitMethod** ↑

Sets the method used for finding the axis limits, and y-ticks.

- > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
- > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
- > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
- > 4 : Uses the MATLAB algorithm for finding the axis limits.

Inherited from superclass NB_GRAPH

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_GRAPH

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_GRAPH

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_GRAPH

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : Don't use this property in combination with one of the graph styles defined by this class. I.e. 'mpr', 'mpr_white', 'presentation' and 'presentation_white' They will set this property to 'normalized'.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_GRAPH

- **graphStyle** ↑

Sets the graphing style of the plots. Give 'mpr' if you want the MPR looking style, or 'presentation' for graphs for the presentation (see nb_Presentation). If you don't want the grey shaded background you can use 'mpr_white' and 'presentation_white' instead.

You can also add your own graph style setting through a .m file. I.e. give the filename (without extension) as a string.

E.g. In the file you can type in something like:

```
set(obj,'titleFontSize',12,'legInterpreter','tex');
```

Caution: If you give this as the first input to the set method it is possible to overrun some of the default settings if that is wanted. E.g. font size of the text, the shading of the plot and so on.

Inherited from superclass NB_GRAPH

- **grid** ↑

Set it to 'on' if grid lines are wanted otherwise set it to 'off'. Default is 'off'.

Inherited from superclass NB_GRAPH

- **gridLineStyle** ↑

Sets the line style of the grid lines. Either '-' , '--' , ':' or '-.' .

Inherited from superclass NB_GRAPH

- **highlight** ↑

If highlighted area between two observations of the graph is wanted set this property.

One highlighted area:

A cell array on the form {obs1, obs2}

More highlighted areas:

A nested cell array on the form
{ {obs1(1), obs2(1)}, {obs1(2), obs2(2)}, ... }

- **highlightColor** ↑

Sets the colors of the background patches. Must be one of the following:

- > A string with the color name to use when plotting (all) the patch(es)
- > A cell array of strings with the color names
- > A cell with 1 x 3 doubles with the RGB color specification.

When given as a cell this property must match the highlight property. I.e. the number of highlighted areas added.

The default color of the highlighted areas is 'light blue'.

- **horizontalLine** ↑

If you want to include some extra horizontal lines, beside the base line, you can set this property. Must be set to a scalar or a double vector with the y-axis value(s) on where to place the horizontal line(s).

Inherited from superclass NB_GRAPH

- **horizontalLineColor** ↑

Sets the color(s) of the given horizontal line(s).

Must either be an M x 3 double with the RGB color(s) or a 1 x M cell array of strings with the color name(s). Where M must be less than the number of plotted horizontal lines.

If less or no color(s) is/are set by this property the default

colors for the rest or all the horizontal line(s) is/are [0 0 0], i.e. MATLAB black.

Inherited from superclass NB_GRAPH

- **horizontalLineStyle** ↑

Sets the style(s) of the given horizontal line(s).

Must either be a string or a 1 × M cell array of strings with the style(s). Where M must be less than the number of plotted horizontal lines.

If less or no style(s) is/are set by this property the default lines for the rest or all the horizontal line(s) is/are '-'.

Inherited from superclass NB_GRAPH

- **horizontalLineWidth** ↑

Sets the line width of the horizontal line(s). Must be a scalar. Default is 1.

Inherited from superclass NB_GRAPH

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_GRAPH

- **legAuto** ↑

'on' | {'off'}. As a string.

> 'on' : The strings given through the fakeLegend and patch properties are automatically added to the legend.

> 'off' : The strings given through the fakeLegend and patch properties are not automatically added to the legend. Which means that you must provide all the legend information through the legends property. The patch descriptions must be given first and the fake legend description must be given last in the legends property. Default.

Inherited from superclass NB_GRAPH

- **legBox** ↑

Set if a box should be drawn around the legend. Either {'on'} | 'off'.

Caution : When removing the box you make it impossible to move it around afterwards.

Inherited from superclass NB_GRAPH

- **legColor** ↑

Sets the color of the background of the box of the legend.
Either as 1x3 double with the RGB or as a string).
Default is 'none', i.e. transparent.

Inherited from superclass NB_GRAPH

- **legColumnWidth** ↑

Sets the width of the columns of the legend. Default is to let MATLAB find the width itself. If given it must be a scalar with the width of all columns of the legend or a 1 x legColumns double with the width of each individual column.

Should be between 0 and 1. Use the try and fail method to find out what fits.

Inherited from superclass NB_GRAPH

- **legColumns** ↑

Sets the number of columns of the legends. Must be a scalar.

Inherited from superclass NB_GRAPH

- **legFontColor** ↑

Sets the font color(s) of the legend text. Must either be a 1x3 double or a string with the color name of all legend text objects, or a M x 3 double or a cellstr array with size 1 x M with color names to use of each legend text object.

Inherited from superclass NB_GRAPH

- **legFontSize** ↑

Sets the font size of the legend. Must be a scalar default is 10.

Inherited from superclass NB_GRAPH

- **legInterpreter** ↑

Sets the interpreter of the legend text. Must be a string.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **legLocation** ↑

Sets the location of the legend, must be string. The location can be:

> 'Best' : Same as 'NorthWest'.
> 'NorthWest'
> 'North'
> 'NorthEast'

```
> 'SouthEast'  
> 'South'  
> 'SouthWest'  
> 'below'      : Below the plot (when many subplots, it will  
                  place the legend below all of them)  
> 'middle'     : Could only be used for 2 x 2 subplot, and then  
                  it places the legend between the upper and  
                  lower subplots.
```

Location that only works for the graph() method:

```
> 'outsideright'   : Legend is placed outside the axes on the  
                     right side, and the axes is resized, so  
                     the axes and legend does not take up more  
                     space than the axes did in the first place.  
                     The legend is vertically centered.  
> 'outsiderighttop' : Same as 'outsideright', but placed  
                     vertically top.
```

Inherited from superclass NB_GRAPH

- **legPosition** [↑](#)

Sets the position of the legend. Must be a 1x2 double. Where
the first element is the x-axis location and the second is the
y-axis location. [xPosition yPosition]

Caution : Both elements must be between 0 and 1. And where we
have that [0, 0] will be the bottom left location of
the axes and [1, 1] is the top right position of the
axes. (This position will be the top left
position of the legend itself.)

Caution : This option overruns the legLocation property.

Inherited from superclass NB_GRAPH

- **legReorder** [↑](#)

Set this property to reorder the legend. Either a string
with {'default'} | 'inverse' or a double with how to reorder
the legend. E.g. if you have 3 legends to plot the default
index will be [1,2,3], the inverse ('inv') will be [3,2,1].
If you want to decide that the 3. legend should be first, then
the 1. and 2., you can type in [3,1,2].

Caution : If a double is given it must have as many elements
as there are plotted legends. I.e. if you give a
empty string to one of the legends it will be
excluded. Say you provide the property
legends as {'s','t','f'} then the double
must have size 1x3.

Inherited from superclass NB_GRAPH

- **legSpace** [↑](#)

The vertical space between the texts of the legend. Must be a
scalar. Default is 0.

Inherited from superclass NB_GRAPH

- **legendText** ↑

Sets the legend(s) of the given variable(s). Must be a cell array on the form: {'var1', 'Name', 'var2', 'Name2', ...}. Here we set the legends of the variables 'var1' and 'var2' to 'Name' and 'Name2' respectively. (The rest will be given the variable names as default).

To give a legend to a splitted line (second part) give;
{...,'Var1(second)', 'LegendSplitted', ...} (Is case sensitiv)

Caution : legAuto should be set to 'on'. Which is not default.
This is only important if the patch or fakeLegend property is used.

Caution : When given this property will overwrite the legends property

Caution : If the variable, scattergroup or candle is not found no error will occure!

Caution : If you have used the patch or fakeLegend property these can be translated as well with this property

Inherited from superclass NB_GRAPH

- **legends** ↑

Give the legends of the plot. Must be given as a cell array of strings.

Default (it depends on the method you use):

> 'graph' : The variable names are used as the default legends. (Given by the variableToPlot and variableToPlotRight properties)

> 'graphSubPlot' : The dataset names are used as the default legends. (Given by the dataNames of the DB property.)

> 'graphInfoStruct' : The dataset names are used as the default legends. (Given by the dataNames of the DB property.)

Caution : The ordering of the legends is as follows:

- patch : See the patch property (only when legAuto is set to 'off')
- variablesToPlot : Then all the variablesToPlot variables. When You have splitted lines, i.e. either using the dashedLine property or the lineStyles property, these can be set after all the variablesToPlot. Have in mind that the ordering is kept.
- variablesToPlotRight : Then comes all the variables

provided by the variablesToPlotRight.
Then the splitted lines.

- fakeLegend : Then you can provide the legend text for the fake legend. (only when legAuto is set to 'off')

Example:

```
data      = nb_ts.rand('2012',10,2);
plotter = nb_graph_ts(data);
plotter.set('variablesToPlot',{'Var1'},...
            'variablesToPlotRight',{'Var2'},...
            'lineStyles',{'Var1',{'-','2014','--'},...
                         'Var2',{'-','2014','--'},...
            'legends',{'Test','Test (--)','Test2','Test2 (--)'});
plotter.graph()
```

Inherited from superclass NB_GRAPH

- **lineStop** ↑

The period that the line should be stopped. E.g. 2. This will trigger the plot type line with bars at end. See also the barPeriods property. Default is [].

- **lineStyles** ↑

Sets the line styles of the given variable(s). Must be a cell array on the form: {'var1', '- ', 'var2', '---', ...}. Here we set the line styles of the variables 'var1' and 'var2' to '-' and '---' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Supported line styles are '-','--','::','.-','---

Caution : For nb_graph_ts and nb_graph_data it is possible to split the plotted line using the following syntax;
{'var1',{'-','2000Q1','--'}} and {'var1',{'-',2,'--'}} respectively

Inherited from superclass NB_GRAPH

- **lineWidth** ↑

Sets the line width of the line plots. Must be a scalar.
Default is 2.5.

Inherited from superclass NB_GRAPH

- **lineWidths** ↑

Sets the line widths of the given variables. Must be a cell array on the form: {'var1', 1, 'var2', 1.5,...}. Here we set the line widths of the variables 'var1' and 'var2' to 1 and 1.5 respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Inherited from superclass NB_GRAPH

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_GRAPH

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_GRAPH

- **mYLim** ↑

Sets the max/min y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty this setting only sets the left y-axis limits.

Only an option for the graphInfoStruct(...) method.

- **markerSize** ↑

Sets the size of the all markers. Must be a scalar. Default is 8.

Inherited from superclass NB_GRAPH

- **markers** ↑

Sets the markers of the given variables. Must be a cell array on the form: {'var1', '^', 'var2', 'o', ...}. Here we set the markers of the variables 'var1' and 'var2' to '^' and 'o' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the table below on which markers that are supported.

'+'	: Plus sign
'o'	: Circle
'*'	: Asterisk
'.'	: Point
'x'	: Cross

```

's' : Square
'd' : Diamond
'^' : Upward-pointing triangle
've' : Downward-pointing triangle
'>' : Right-pointing triangle
'<' : Left-pointing triangle
'p' : Five-pointed star (pentagram)
'h' : Six-pointed star (hexagram)

Inherited from superclass NB_GRAPH

- missingValues ↑

Set how to interpret missing observations. Must be a string.
Either;

> 'interpolate' : Linearly interpolate the missing values.

> 'none'           : No interpolation (default)

- nanAfterObs ↑

If setted this property would make the graph blank after the
provided obs, but the x-axis would still keeps its axis
limits. Must be an integer.

Caution: Not possible to combine with variablesToPlotX

- nanBeforeObs ↑

If setted this property would make the graph blank before the
provided obs, but the x-axis would still keeps its axis
limits. Must be an integer.

Caution: Not possible to combine with variablesToPlotX

- nanVariables ↑

Sets the nan option for individual variables. Must be given
as a cell. I.e. {'var1', {2, 4}, ...
'var2', {2, 4}, ...}. This will make both 'var1' and 'var2'
variables blank before 2 and after 4 (Not including.)
in the given graph.

Caution: Not possible to combine with variablesToPlotX

- noLabel ↑

Set it to 1 if no label(s) is/are wanted on the graphs,
otherwise set it to 0. Both on the y-axis and x-axis.
Default is 0. This is of course only have an effect if the
nb_graph object have been given the xLabel or yLabel
properties. (These are empty by default.)

Inherited from superclass NB_GRAPH

- noLegend ↑

```

Set it to 1 if no legend is wanted, otherwise set it to 0.
Default is 0.

Inherited from superclass NB_GRAPH

- **noTickMarkLabels** ↑

Set to true (1) to remove tick marks and tick marks label of axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsLeft** ↑

Set to true (1) to remove tick marks and tick marks label of the left side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsRight** ↑

Set to true (1) to remove tick marks and tick marks label of the right side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the graphs, otherwise set it to 0. Default is 0.

Set it to 2 if you want the dataNames{ii} as the title of the graph when using the graph method.

Inherited from superclass NB_GRAPH

- **normalized** ↑

If the font should be normalized to the figure or axes.
Either 'figure' or 'axes'.

Inherited from superclass NB_GRAPH

- **numberOfGraphs** ↑

Number of graph produced by the graph methods. Not settable.

Inherited from superclass NB_GRAPH

- **obs** ↑

All the dates of the plot as a cellstr. Not settable.

- **page** ↑

Sets the page of the data object to plot. Only an option for graph() method. Default is [], i.e. plot all pages in separate figures.

Inherited from superclass NB_GRAPH

- **parameters** ↑

A struct with the parameters that can be used in evaluation of expressions in the graphSubPlots and graphInfoStruct methods.

Inherited from superclass NB_GRAPH

- **patch** ↑

Sets the patch property of the object. If set this will add a patch (color fill) between two variables.

Only one patch :

```
{'patchName','var1','var2',color}
```

More patches :

```
{'patchName1','var1Patch1','var2Patch1',color1 ,...  
'patchName2','var1Patch2','var2Patch2',color2,...}
```

Where the the color option(s) must either be a 1 x 3 double with the RGB colors or a string with the color name.

The first input will be the legend text of the patch. I.e. 'patchName', 'patchName1' and 'patchName2' will be the description text included in the legend. (If not the property legAuto is set to 'off'.)

Inherited from superclass NB_GRAPH

- **patchAlpha** ↑

Sets the transparency of the patches. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_GRAPH

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_GRAPH

- **plotType** ↑

Sets the plot type. Either:

```
> 'line'           : If line plot(s) is wanted. Default.  
> 'stacked'       : If stacked bar plot(s) is wanted. Not an  
                     option for the graphSubPlots(...) method.  
> 'grouped'       : If grouped bar plot(s) is wanted.  
> 'dec'            : If decomposition plot(s) is wanted. Which is  
                     a bar plot with also a line with the sum of  
                     the stacked bars. Not an option for the  
                     graphSubPlots(...) method.  
> 'area'           : If area plot(s) is wanted.  
> 'candle'          : If candle plot(s) is wanted.  
> 'scatter'         : If scatter plot(s) is wanted.
```

- **plotTypes** ↑

Sets the plot type of the given variables. Must be a cell array on the form: {'var1', 'line', 'var2', 'grouped', ...}. Here we set the plot type of the variables 'var1' and 'var2' to 'line' and 'grouped' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the property plotType above on which plot types that are supported.

- **position** ↑

Sets the position of the axes, must be an 1x4 double. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

For the graph method graphSubPlots the position input can be given a 1 x N cell array, where each element is a 1x4 double. N = obj.subPlotSize(1)*obj.subPlotSize(2).

Inherited from superclass NB_GRAPH

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

Inherited from superclass NB_GRAPH

- **scatterLineStyle** ↑

Sets the scatter line style. Must be string. See 'lineStyles' for supported inputs. (Sets the line style of all scatter plots)

- **scatterObs** ↑

Sets the start and end to be used for the scatter plot plotted against the left axes. Must be a nested cell array. The obs can be given as integers. E.g:

```
{'scatterGroup1',{'startObs1','endObs1'},...  
'scatterGroup2',{'startObs2','endObs2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterObsRight)

Be aware : Each obs will result in one point in the scatter.

- **scatterObsRight** ↑

Sets the start and end to be used for the scatter plot plotted against the right axes. Must be a nested cell array. The obs can be given as integers. E.g:

```
{'scatterGroup1',{'startObs1','endObs1'},...  
'scatterGroup2',{'startObs2','endObs2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterObs)

Be aware : Each obs will result in one point in the scatter.

- **scatterVariables** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the left y-axis. E.g:

```
{'var1','var2'}
```

- **scatterVariablesRight** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the right y-axis. E.g:

```
{'var1','var2'}
```

- **shading** ↑

The shading option of the axes background:

- 'grey' : Shaded grey background
- 'none' : Background color given by the color property.

- A $n \times m \times 3$ double with the background color. n is the number of horizontal pixels, m is the number of vertical pixels and 3 means the RGB colors.

Inherited from superclass NB_GRAPH

- **spacing** ↑

Sets the x-ticks spacing. Must be an integer (or a double when variableToPlotX is used)

- **startGraph** ↑

Sets the start obs of the graph. Must be a string or an integer. If this obs is before the start obs of the data, the data will be expanded with nan (blank values) for these periods.

Not used when variableToPlotX property is set, use xLim instead.

- **subPlotSize** ↑

Sets the number of subplots per figure. Must be a 1×2 double with the number of subplot rows as the first element and the number of subplot columns as the second element. Only an option of the graphSubPlots() and graphInfoStruct() methods.

Default is [2, 2].

Inherited from superclass NB_GRAPH

- **subPlotSpecial** ↑

Set it to 1 if you want the 1×2 and 2×1 subplots to better fit for usage in presentations. Default is 0, i.e. use MATLAB default positions.

When set to 1 this class uses the nb_subplotSpecial instead of nb_subplot to find the positions of the subplots.

Inherited from superclass NB_GRAPH

- **sumTo** ↑

Sets a number which the area or the stacked bar plot should sum to. Must be a scalar. I.e. if 100 is given. The sizes of the bars will be the percentage share of the total sum.

Inherited from superclass NB_GRAPH

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_GRAPH

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_GRAPH

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_GRAPH

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_GRAPH

- **variableToPlotX** ↑

Set the variable to use as the x-axes variable. I.e. when you want to plot one or more variables against this one. As a string.

If the graphSubPlots method is used this input can be set to cellstr with the same size as the variablesToPlot input.
You can use this to set individual domains for each subplot.

- **variablesToPlot** ↑

Sets the variables to plot against the left (or both) axes.
Must be a cell array of strings with the variable names. I.e.
{'var1', 'var2', ...}.

Not an option for the graphInfoStruct(...), which use the
property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **variablesToPlotRight** ↑

Sets the variables to plot against the right axes. Must be
a cell array of strings with the variable names. I.e.
{'var1', 'var2', ...}.

Not an option for the graphInfoStruct(...), which use the
property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **verticalLine** ↑

Sets the obs(s) of where to place a vertical line(s).
(Dotted orange line is default). Must be a number or a cell
array of numbers with the obs(s) for where to place the
vertical line(s). I.e. obs1 or {obs1,obs2,...}

- **verticalLineColor** ↑

Sets the color(s) of the given vertical line(s). Must either
be an M x 3 double with the RGB color(s) or a 1 x M cell array
of strings with the color name(s). Where M must be less than
the number of plotted vertical lines.

If less or no color(s) is/are set by this property the default
color(s) for the rest or all the vertical line(s) is/are
[0 0 0]. I.e. MATLAB black.

- **verticalLineLimit** ↑

Sets the y-axis limit(s) of the given vertical line(s). Must
either be a 1 x M cell array of 1 x 2 double with the upper
and lower y-axis limit(s). Where M must be less than the
number of plotted vertical lines.

If less or no limit(s) is/are set by this property the default
limit for the rest or all the vertical line(s) is/are the full
height of the graph.

- **verticalLineStyle** ↑

Sets the line style(s) of the given vertical line(s). Must
either be a 1 x M cell array of strings with the style(s).
Where M must be less than the number of plotted vertical
lines.

If less or no style(s) is/are set by this property the default line style is/are used for the rest or all the vertical line(s) is/are '--'.

- **verticalLineWidth** ↑

Sets the line width of (all) the vertical line(s). Must be a scalar. Default is 1.5.

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar. Default is 12.

Inherited from superclass NB_GRAPH

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string. Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **xLim** ↑

Sets the x-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Only an option when the plotType property is set to 'scatter' or when variableToPlotX is used.

- **xScale** ↑

Sets the scale of the x-axis. Either {'linear'} | 'log'.

Only an option when plotType is set to 'scatter' or when variableToPlotX is used.

- **xTickLabelAlignment** ↑

The alignment of the x-axis tick marks labels. {'normal'} | 'middle'.

- **xTickLabelLocation** ↑

The location of the x-axis tick marks labels. Either {'bottom'} | 'top' | 'baseline'. 'baseline' will only work in combination with the xtickLocation property set to a double with the basevalue.

- **xTickLabels** ↑

Sets the x-axis tick mark labels. To translate the default tick mark label 'Var1' and 'Var2' you can use;

```
{'Var1','Label1','Var2','Label2'}
```

If given as a empty cell, default x-axis tick marks will be used, which is default.

This property can be used to create multi-lined x-axis tick mark labels. To do this give a multi-row char to the labels you want to make multi-lined.

Inherited from superclass NB_GRAPH

- **xTickLocation** ↑

The location of the x-axis tick marks. Either {'bottom'} | 'top' | double. If given as a double the tick marks will be placed at this value (where the number represent the y-axis value).

- **xTickRotation** ↑

Sets the rotation of the x-axis tick mark labels. Must be a scalar with the number of degrees the labels should be rotated. Positive angles cause counterclockwise rotation.

- **xTickStart** ↑

Sets the start obs of where to start the x-tick mark labels.
Must be an integer (a double when variableToPlotX is used).

- **yDir** ↑

Sets the direction of the y-axis of the plot. Must be a string. Either 'normal' or 'reverse'. Default is 'normal'.

Caution: If the variablesToPlotRight is not empty this setting only sets the left y-axis direction.

- **yDirRight** ↑

Sets the direction of the right y-axis of the plot. Either 'normal' or 'reverse'. Default is 'normal'. (Only when the variablesToPlotRight property is not empty.)

- **yLabel** ↑

Sets (add) the text of the y-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **yLabelFontSize** ↑

Sets the font size of the y-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_GRAPH

- **yLabelFontWeight** ↑

Sets the font weight of the y-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **yLabelInterpreter** ↑

The interpreter used for the given string given by the xLabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **yLabelRight** ↑

Sets the text of the y-axis label. (Only on the right side.)
Must be a string. Takes the same font option as the yLabel property

Inherited from superclass NB_GRAPH

- **yLim** ↑

Sets the y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty this setting only sets the left y-axis limits.

Only an option for the graph(...) and graphSubPlots() methods. (For the graphInfoStruct(...) method use the property GraphStruct.)

- **yLimRight** ↑

Sets the right y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Only when the variablesToPlotRight property is not empty.) (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Only an option for the graph(...) and graphSubPlots() methods. (For the graphInfoStruct(...) method use the property GraphStruct.)

- **yOffset** ↑

The y-axis tick mark labels offset from the axes

- **yScale** ↑

Sets the scale of the y-axis. {'linear'} | 'log'. (Both the left and the right axes if nothing is plotted on the right axes.)

- **yScaleRight** ↑

Sets the scale of the right y-axis. {'linear'} | 'log'. (Has only something to say if something is plotted on the right axes)

- **ySpacing** ↑

Sets the spacing between y-axis tick mark labels of the y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow the frequency of the xTickFrequency property if setted, else it will follow the frequency of the data provided. (Except when dealing with daily data, because then the default xTickFrequency is 'yearly')

Caution: If the variablesToPlotRight property is not empty this property only sets the spacing between ticks of the left y-axis.

- **ySpacingRight** ↑

Sets the spacing between y-axis tick mark labels of the right y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow

Methods:

- | | | | |
|-----------------------------------|-------------------------------------|---------------------------------------|------------------------------|
| • createFigure | • get | • getCode | • getData |
| • getDataOptions | • getGenericOptions | • getPlottedVariables | • graph |
| • graphInfoStruct | • graphSubPlots | • isempty | • merge |
| • realTimespan | • resetDataSource | • saveData | • saveFigure |
| • set | • struct | • timespan | • update |

► **createFigure** ↑

```
createFigure(obj)
```

Description:

Create figure to plot in.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_graph_data

Input:

- obj : An object of class nb_graph_data
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_graph_data object. Be aware that all the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_graph_data object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#), [nb_data](#), [nb_graph](#)

Written by Kenneth S. Paulsen

► **getCode** ↑

```
[code,numOfGraphs] = getCode(obj,frameTitle,frameSubTitle,...  
                               footer,source,pageNumber,theme)
```

Description:

Get the latex code when including a graph in a slide object produced by an object of class nb_graph_ts.

This method is called from the nb_Slide class, when making a beamer slide in latex.

Input:

- obj : An object of class nb_graph_ts
- frameTitle : A string with the frame title of the slide.
- frameSubTitle : A string with the frame subtitle of the slide.
- footer : A cellstr with the footers of the slide
- source : A cellstr with the sources of the slide
- pageNumber : Give 1 if the page number should be included.
- theme : The beamer theme used. Default is '' (use default beamer theme).

Output:

- code : The latex code as a char.
- numOfGraphs : The number of graphs produced by the input object of class nb_graph_ts. As an integer.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **getData** ↑

```
data = getData(obj)
```

Description:

Get the data of the graph

Input:

- obj : An object of class nb_graph_data

Output:

- data : As an nb_data object

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

► **getDataOptions** ↑

```
options = getDataOptions(obj)
```

Description:

Container with the properties specific to nb_graph_data

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getDataOptions(obj);
```

See also:

[set](#), [getGenericOptions](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

► **getGenericOptions** ↑

```
options = getGenericOptions(obj)
```

Description:

A static method of nb_graph.

Container with the intersect of options for subclasses of nb_graph.

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getGenericOptions(obj);
```

See also:

[set](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the plotted variables of the graph

Input:

- obj : An object of class nb_graph_data
- forLegend : true or false (default)

When forLegend is used the variables are listed as needed for the legend, but without fake legends and patches, and duplicates are not removes.

Output:

- vars : As a cellstr

Example:

```
vars = obj.getPlottedVariables();
```

Written by Kenneth S. Paulsen

► **graph ↑**

```
graph(obj)
```

Description:

Create a graph or more graphs

Graphs all the variables given by the variablesToPlot and the variablesToPlotRight properties in one plot, and does that for all the datasets of the DB property. I.e. one new plot for each dataset (page) of the DB property of the nb_graph_data object.

Input:

- obj : An object of class nb_graph_data

Output:

The graph plotted on the screen.

Examples:

```
data = nb_data([2;1;2],'',1,'Var1');  
obj = nb_graph_ts(data);  
obj.graph();
```

See also:

[nb_graph_data](#), [nb_graph_data.set](#)

Written by Kenneth S. Paulsen

► graphInfoStruct ↑

```
graphInfoStruct(obj)
```

Description:

Create graphs based on the property GraphStruct. (Or the default GraphStruct given by the method defaultGraphStruct)

This method makes it easy to set up graph packages. For example graphing output from models.

See the documentation NB toolbox for more on this method.

Input:

- obj : An object of class nb_graph_data, where the property GraphStruct contains the information on how to plot the objects data.

Output:

The wanted graphs plotted on the screen.

Examples:

```
obj.graphInfoStruct();
```

Written by Kenneth S. Paulsen

► graphSubPlots ↑

```
graphSubPlots(obj)
```

Description:

Create graphs of all the variables given in variablesToPlot property, in different subplots. (Each dataset against each other). The number of subplot is given by the property subPlotSize.

Input:

- obj : An object of class nb_graph_data

Output:

The wanted graphs plotted on the screen.

Examples:

```
data = nb_data([2 3 2;1 4 5;2 3 2],'',1,...  
              {'Var1','Var2','Var3'});  
obj = nb_graph_data(data);  
obj.graphSubPlots();
```

Written by Kenneth S. Paulsen

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_graph_data object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_graph_data

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► **merge** ↑

```
out = merge(obj,varargin)
```

Description:

Default: This tries to merge the data of plotter objects that uses the graphSubPlots, or the graphInfoStruct.

Caution: All objects must share the exact same variables!

'graph' is given: In this case it will try to merge the data of all nb_graph objects, and concatenate the variableToPlot property. If the property variableToPlot is empty for all objects, it will also be returned as empty, i.e. plot all series in the data.

Caution : Merging of any of the other properties then DB and variableToPlot is not done!

Input:

- obj : An object of class nb_graph, which includes the classes nb_graph_ts, nb_graph_cs and nb_graph_data.

Optional input:

- 'graph' : Give this input to merge the graphs in the way that is described in the description of this method.
- varargin : Each input must be an object of class nb_graph.

Output:

- out : An object of class nb_graph.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► realTimespan ↑

```
dateString = realTimespan(obj)
```

Description:

Get the timespan of the data behind a graph which is not nan or infinite, as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_data
- language : Has nothing to say

Output :

- obsString : A string with the span of the graph represented by this object.

Examples:

```
data      = nb_data([2;1;2],',','1','Var1');
obj      = nb_graph_data(data);
dateString = realTimespan(obj);
```

Written by Kenneth S. Paulsen

► resetDataSource ↑

```
[message,err] = resetDataSource(obj,varargin)
```

Description:

Reset the data source of the nb_graph_data object.

Input:

- obj : An object of class nb_graph_data
- dataSource : An nb_data object with the new data.
- updateProps : If the variablesToPlot should be updated to be equal to dataSource.variables + the startGraph and endGraph properties are set to the startDate and endDate of the new data source.

Can also be 'gui'. Then all properties will be checked against the new dataset, and if the new dataset does not comply with the properties, the properties will be updated!

Written by Kenneth S. Paulsen

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_graph_data
- filename : A string with the saved output name
- strip : - 'on' : Strip all observation dates where all the variables has no value. Default.
- 'off' : Does not strip all observation dates where all the variables has no value

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_graph
- extraName : If you want to add some extra name to the saveName property of the nb_graph_data object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Examples:

```
data = nb_data([2;1;2],'', '1', 'Var1');  
obj = nb_graph_data(data);  
obj.graph();  
saveFigure(obj,'test','pdf');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **set ↑**

```
set(obj,varargin)
```

Description:

Set properties of the nb_graph class and its subclasses.

See documentation NB Toolbox or type help('nb_graph') for more on the properties of the class.

Input:

- obj : An object that is a subclass of nb_graph
- varargin : 'propertyName',PropertyValue,...

Output:

No actual output, but the input objects properties will have been set to their new value.

Examples:

```
data = nb_ts([2;1;2],'', '2012Q1', 'Var1');
obj = nb_graph_ts(data);
obj.set('startGraph', '2012Q1', 'endGraph', '2012Q3');
obj.set('crop', 1);
```

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► struct ↑

```
s = struct(obj)
```

Description:

Convert object to struct

Input:

- obj : An object of class nb_graph_cs

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► timespan ↑

```
dateString = timespan(obj, language)
```

Description:

Get the timespan of the graph as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_data
- language : Has nothing to say

Output :

- dateString : A string with the span of the graph represented by this object.

Examples:

```
data      = nb_data([2;1;2],'',1,'Var1');
obj      = nb_graph_data(data);
dateString = timespan(obj);
```

Written by Kenneth S. Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the figure

See the update method of the nb_data for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_graph_data

Output:

- obj : An object of class nb_graph_data, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ **nb_graph_package**
Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_package(start,chapter)
```

Superclasses:

handle

Description:

Class for creating a graph package (MPR styled graphs)

Constructor:

```
obj = nb_graph_package(start,chapter)
```

Input:

- start : The start number of the package. As a scalar.
- chapter : The chapter of the package. As a scalar.

Output:

- obj : An object of class nb_graph_package

Examples:

Start with the first graph of the chapter 1.

```
obj = nb_graph_package(1,1);
```

See also:

[nb_graph_adv](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- a4Portrait
- advanced
- bigLetter
- chapter
- excelStyle
- firstPageNameEng
- firstPageNameNor
- flip
- graphs
- identifiers
- lookUpMatrix
- roundoff
- start
- userData
- zeroLowerBound

- **a4Portrait** [↑](#)

Save PDF to A4 portrait format. 0 or 1

- **advanced** [↑](#)

Set the window size to fit the advanced graphs (i.e. nb_graph_adv objects). Default is true. When set to false the normally produced graphs will be on the correct scale (i.e. nb_graph_ts, nb_graph_cs, nb_graph_data and nb_graph_subplot)

- **bigLetter** [↑](#)

Give 1 the numbering should be done with a big letter instead of a number. I.e. 'Figur 1.A' or 'Chart 1.A', instead of 'Figure 1.1' or 'Chart 1.1'.

- **chapter** [↑](#)

The chapter of the graph package. As an integer.

- **excelStyle** [↑](#)

Sets the style of the Excel sheet the data is being exported into using saveData.
Options are MPR (Monetary Policy Rapport) and FSR (Financial Stability Rapport). Default is MPR.

- **firstPageNameEng** ↑

Sets the heading on the index page of the excel spreadsheet. When the language input to the method saveData is 'english' or 'both' this will be the heading of the english index page (Or else this property does nothing). Must be given as a string or as an cellstr. A cellstr will give a multilined heading. E.g. 'Index page' or {'Index page','second line'}. Default is empty.

- **firstPageNameNor** ↑

Sets the heading on the index page of the excel spreadsheet. When the language input to the method saveData is 'norwegian' or 'both' this will be the heading of the norwegian index page. Must be given as a string or as an cellstr. A cellstr will give a multilined heading. E.g. 'Index page' or {'Index page','second line'}. Default is empty.

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

- **graphs** ↑

All the graphs of the package, as a cell of nb_graph_adv and nb_graph_subplot objects.

- **identifiers** ↑

A cellstr with the graph objects identifiers

- **lookUpMatrix** ↑

A cell matrix on how to translate the mnemonics to variable description in the excel output spreadsheet. Both in english and norwegian. Use the set method of this class for setting this property.

Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {

'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

- **roundoff** ↑

Give the number of decimals of the excel output produced by the saveData method. Default is 2. Must be an integer greater or equal to 0, or empty (no round-off).

- **start** ↑

The starting number of the figure package. As an integer.

- **userData** ↑

User data, can be set to anything you want.

- **zeroLowerBound** ↑

Zero lower bound QUA_RNFOLIO and QUA_RNFOLIO_LATESTMPR

Methods :

- add
- getSupportedTemplates
- moveDown
- moveLast
- preview
- remove
- set
- update
- writePDFExtended
- writeSeparateExtended
- exportExcelChangeLog
- importExcelChangeLog
- moveFirst
- moveUp
- previewExtended
- saveData
- switchLocation
- writePDF
- writeSeparate
- writeText

► **add** ↑

add(obj, graph, identifier)

Description:

Add a graph to package. Must be an object of class nb_graph_adv or nb_graph_subplot.

Input:

- obj : An object of class nb_graph_package
- graph : An object of class nb_graph_adv or nb_graph_subplot

```
- identifier : A string to link to the object. Used by the
    reorder, remove and reset methods.

    If given as empty, a default identifier is given.
    I.e. 'GraphX'. Where X is the already added graph
    + 1.

    Caution : The identifier must be unique.
```

Examples:

```
obj.add(nb_graph_advObj)
obj.add(nb_graph_advObj,'Test')
```

Written by Kenneth S. Paulsen

► **exportExcelChangeLog** ↑

```
exportExcelChangeLog(obj,saveName,packageName)
```

Description:

Exports the written information in a graph package to an Excel file.
The exported file will contain titles, footers, and excel footers with
all DAG formatting visible.
Changes can be made to the exported file. This file can be imported to
DAG and all changes will take place.

Input:

- obj : An object of class nb_graph_package.
- saveName : The name of the Excel file to write.
- packageName : The name of the graph package.

Examples:

```
obj.writeText('MyGraphPackage');
```

Written by Per Bjarne Bye

► **getSupportedTemplates** ↑

```
templates = getSupportedTemplates(obj)
```

Description:

Get the the templates that are shared among the graph. If some graphs does not have an assigned template the output will be {}.

Input:

- obj : An object of class nb_graph_package.

Output:

- templates : A cellstr with the shared templates.

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **importExcelChangeLog** ↑

`exportExcelChangeLog(obj,excelFile)`

Description:

Reads the Excel file and updates the written information in the graph package based on the Excel file. Note that the file must be structured such that it is identical to a file created by `exportExcelChangeLog`.

Input:

- obj : An object of class nb_graph_package.
- excelFile : 1 x n char. The Excel file with the information.

Examples:

`obj.importExcelChangeLog('myExcelFile.xls');`

Written by Per Bjarne Bye

► **moveDown** ↑

`moveDown(obj,identifier)`

Description:

Moves the graph stored with the identifier provided 1 step down in the package. (If it is the last nothing will happen.)

Input:

- obj : An object of class nb_graph_package.
- identifier : A String with the identifier to the graph to move down one step.

Examples:

```
obj.moveDown('Graph1')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **moveFirst** ↑

```
moveFirst(obj, identifier)
```

Description:

Moves the graph stored with the identifier provided first in the package.

Input:

- obj : An object of class nb_graph_package.
- identifier : A String with the identifier to the graph to move last.

Examples:

```
obj.moveFirst('Graph1')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **moveLast** ↑

```
moveLast(obj, identifier)
```

Description:

Moves the graph stored with the identifier provided last in the package.

Input:

- obj : An object of class nb_graph_package.
- identifier : A String with the identifier to the graph to move last.

Examples:

```
obj.moveLast('Graph1')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **moveUp** ↑

```
moveUp(obj,identifier)
```

Description:

Moves the graph stored with the identifier provided 1 step up in the package. (If it is the first nothing will happen.)

Input:

- obj : An object of class nb_graph_package.
- identifier : A String with the identifier to the graph to move up one step.

Examples:

```
obj.moveUp('Graph1')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **preview** ↑

```
preview(obj,language/gui,template)
```

Description:

Preview of the graph package.

Input:

- obj : An object of class nb_graph_package
- language : The language as a string, 'english' or {'norsk'}
- gui : true or false
- template : A one line char with the applied template. Default is 'current', i.e. do specific template.

Examples:

```
obj.preview('english');
```

Written by Kenneth S. Paulsen

► **previewExtended** ↑

```
previewExtended(obj,language/gui)
```

Description:

Preview of the graph package.

Input:

- obj : An object of class nb_graph_package
- language : The language as a string, 'english' or {'norsk'}
- gui : true or false

Examples:

```
obj.previewExtended('english');
```

Written by Kenneth S. Paulsen

► **remove** ↑

```
remove(obj, identifier)
```

Description:

Removes the graph stored with the identifier provided.

Input:

- obj : An object of class nb_graph_package.
- identifier : A String with the identifier to the graph to remove.

Examples:

```
obj.remove('Graph1')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **saveData** ↑

```
saveData(obj, saveName, language)
saveData(obj, saveName, language, firstPageName, firstPageName2, roundoff, ...
         excelStyle)
```

Description:

Save the data behind the graphs of the graph package to excel.

Input:

- obj : An object of class nb_graph_package
- saveName : The excel file name as a string
- language : The language as a string. One of the following:
 - > 'english' : The data are written to a excel spreadsheet with only english text.
 - > 'norsk' : The data are written to a excel spreadsheet with only norwegian text.
 - > 'both' : The data are written to a excell spreadsheet with both norwegian and english text.

- firstPageName : Sets the heading on the index page of the excel spreadsheet. When the language input is 'norwegian' or 'both' this will be the heading of the norwegian index page. Must be given as a string or as an cellstr. A cellstr will give a multilined heading. E.g. 'Index page' or {'Index page','second line'}.
- firstPageName2 : Sets the heading on the index page of the excel spreadsheet. When the language input is 'english' or 'both' this will be the heading of the english index page. Must be given as a string or as an cellstr. A cellstr will give a multilined heading. E.g. 'Index page' or {'Index page','second line'}.
- roundoff : Give 1 if you want to round off to 2 decimals, 0 otherwise. Default is obj.roundoff.

Examples:

```
package.saveData('test','both',{'Rapport',...
    'med noe nyttig'},...
{'A report',...
    'with something useful'},...
1);
```

Written by Per Bjarne Bye and Kenneth S. Paulsen

► **set ↑**

```
set(obj,varargin)
```

Description:

Set the properties of the nb_graph_package object

Input:

- obj : An object of class nb_graph_package
- varargin :,'propertyName',PropertyValue,...

Output :

- obj : The input objects properties are updated.

Examples:

```
obj.set('propertyName',PropertyValue,...)
obj = obj.set('lookupmatrix',{'QUA_PCPI','CPI','KPI'});
```

Written by Kenneth S. Paulsen

► **switchLocation** ↑

```
switchLocation(obj, identifier1, identifier2)
```

Description:

Moves the graph stored with the identifier provided last in the package.

Input:

- obj : An object of class nb_graph_package.
- identifier1 : A String with the identifier to the graph to switched.
- identifier2 : A String with the identifier to the graph to switched.

Examples:

```
obj.switchLocation('Graph1','Graph2')
```

See also:

[nb_graph_package](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the nb_graph_package object. I.e update the data source of the graphs property.

See the update method of the nb_cs or nb_ts class for more on how to make the data of the graphs updateable.

Input:

- obj : An object of class nb_graph_package
- addWaitBar : Add wait bar if 1, otherwise no. Default is not.

Output:

- obj : An object of class nb_graph_package, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

► **writePDF** ↑

```
writePDF(obj,saveName,language/gui,template)
```

Description:

Save the graph package to a pdf file

Input:

- obj : An object of class nb_graph_package
- saveName : The name of the pdf file as a string. If empty no file is made.
- language : The language as a string, 'english' or {'norsk'}
- gui : true or false
- template : A one line char with the applied template. Default is 'current', i.e. do specific template.

Examples:

```
obj.writePDF('test', 'english');
```

Written by Kenneth S. Paulsen

► **writePDFExtended** ↑

```
writePDFExtended(obj,saveName,language/gui)
```

Description:

Save the graph package to a pdf file with more information on the graphs.

Input:

```

- obj : An object of class nb_graph_package

- saveName : The name of the pdf file as a string. If empty no file
is made.

- language : The language as a string, 'english' or
{'norsk'}

- gui : true or false

```

Examples:

```
obj.writePDFExtended('test', 'english');
```

Written by Kenneth S. Paulsen

► **writeSeparate ↑**

```
writeSeparate(obj, folder, language, format, index, gui, template, ...
includeFigureNumber, crop)
```

Description:

Save the graph package to files. Each graph will be save with the name of the assign identifier. See the nb_graph_package.add for more on the identifiers.

Input:

```

- obj : An object of class nb_graph_package

- folder : The folder to save the graphs to.

- language : The language as a string, 'english' or
{'norsk'}. Will also be appended the save names.

- format : The file format to save the graph to. See the nb_saveas
function for the supported formats. Default is 'pdf'.

- index : Either a 1 x numGraphs logical or empty. Set the matching
element of a graph to false to skip printing the specific
graph. Default is empty, i.e. to print all graphs of the
package.

- gui : true or false

- template : The template of the graphs to use. Default is 'current'.

- includeFigureNumber : Include figure number in file names. Default
is false.

- crop : true or false. Default is true.

- skipConfirmation : Skip the confirmation window and go straight to
writing or not. Default is false.

```

Examples:

```
obj.writeSeparate('test', 'english', 'pdf');
```

Written by Kenneth S. Paulsen

► **writeSeparateExtended** ↑

```
writeSeparateExtended(obj, folder, language, format, index, gui, ...  
                      includeFigureNumber)
```

Description:

Save the graph package to files. Each graph will be save with the name of the assign identifier. See the nb_graph_package.add for more on the identifiers. Tooltip, excel title and footer will be added to the graph.

Input:

- obj : An object of class nb_graph_package
- folder : The folder to save the graphs to.
- language : The language as a string, 'english' or {'norsk'}. Will also be appended the save names.
- format : The file format to save the graph to. See the nb_saveas function for the supported formats. Default is 'pdf'.
- index : Either a 1 x numGraphs logical or empty. Set the matching element of a graph to false to skip printing the specific graph. Default is empty, i.e. to print all graphs of the package.
- gui : true or false
- includeFigureNumber : Include figure number in file names. Default is false.
- skipConfirmation : Skip the confirmation window and go straight to writing or not. Default is false.

Examples:

```
obj.writeSeparate('test', 'english', 'pdf');
```

Written by Kenneth S. Paulsen

► **writeText** ↑

```
writeText(obj, folder, language, index, includeFigureNumber)
```

Description:

Write text of graphs to .txt files. One for the figure title and one for the footer for each graph separately. Each graph will be save with the name of the assign identifier. See the nb_graph_package.add for more on the identifiers.

Input:

- obj : An object of class nb_graph_package
- folder : The folder to save the text of the graphs to.
- language : The language as a string, 'english' or {'norsk'}. Will also be appended the save names.
- index : Either a 1 x numGraphs logical or empty. Set the matching element of a graph to false to skip writing the text of that graph. Default is empty, i.e. to write the text of all graphs of the package.
- includeFigureNumber : Include figure number in file names. Default is false.
- singleFolder : Save files to a single folder or make two subfolders and save text to one and extended text to the other. Default is true.
- lazyWriting : Write only the text files that have changed (cannot find the file or the file content is identical to what is about to be written). The files that are written are reported in a separate window. Default is false
- skipConfirmation : Skip the confirmation window and go straight to writing or not. Default is false.

Examples:

```
obj.writeText('test', 'english');
```

Written by Kenneth S. Paulsen

■ nb_graph_subplot

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_subplot(varargin)
```

Superclasses:

handle

Description:

This is a class for making subplots of nb_graph_ts and nb_graph_cs objects (graphs).

Constructor:

```
obj = nb_graph_subplot(varargin)
```

Input:

- varargin : Arbitrary number of objects of class nb_graph_ts or nb_graph_cs.

Output:

- obj : An object of class nb_graph_subplot

See also:

[nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- a4Portrait
- figureNameEng
- fileFormat
- graphStyle
- numberOfGraphs
- saveName
- subPlotSize
- axesScaleLineWidth
- figureNameNor
- flip
- language
- pdfBook
- scale
- subPlotSpecial
- crop
- figurePosition
- graphObjects
- manualAxesPosition
- plotAspectRatio
- shading
- userData

- **a4Portrait** ↑

Save PDF to A4 portrait format. 0 or 1

- **axesScaleLineWidth** ↑

Scale line width to axes height. true or false (default).

- **crop** ↑

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

- **figureNameEng** ↑

Sets the english name of the figure. This name will be used on the englisp index page of the excel spreadsheet.

- **figureNameNor** ↑

Sets the norwegian name of the figure. This name will be used on the norwegian index page of the excel spreadsheet.

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

- **fileFormat** ↑

Sets the save file format. Either 'eps', 'jpg', 'pdf' or 'png'. 'pdf' is default.

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

- **graphObjects** ↑

The graphs to include in the figure as subplots. As nb_graph_ts, nb_graph_data, nb_graph_cs or nb_graph_adv objects.

- **graphStyle** ↑

Will if set to 'presentation' or 'presentation_white' only decide the some figure option. Use the graphStyle property of the added graph objects to set the graph style.

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

- **manualAxesPosition** ↑

Set the axes position to manual. Default is false

- **numberOfGraphs** ↑

Number of graph produced by the graph methods. Not settable

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3.

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

- **scale** ↑

Scale all the font sizes of the plotted graphs of the subplot. (Except nb_annotation objects). Default is 0.7.

- **shading** ↑

Sets the shading of the background. Either 'grey' or 'none'. 'none' will give a white colored background.

- **subPlotSize** ↑

Sets the number of subplots per figure. Must be a 1 x 2 double with the number of subplot rows as the first element and the number of subplot columns as the second element. Only an option of the graphSubPlots() method.

Default is [2, 2].

- **subPlotSpecial** ↑

Set it to 1 if you want the 1x2 and 2x1 subplots to better fit for usage in presentations. Default is 0, i.e. use MATLAB default positions.

When set to 1 this class uses the nb_subplotSpecial instead of nb_subplot to find the positions of the subplots.

- **userData** ↑

User data, can be set to anything you want.

Methods:

- add
 - delete
 - get
 - getCode
 - graph
 - graphEng
 - graphNor
 - remove
 - reset
 - saveData
 - saveFigure
 - set
 - update
-

► add ↑

```
add(obj,graphObj)
```

Description:

Adds a graph object to the subplot object.

Caution : This function will copy the provided object before its adds the object to the nb_graph_subplot object.

Input:

- obj : An object of class nb_graph_subplot
- graphObj : An object of class nb_graph_ts or nb_graph_cs.

Example:

```
obj.add(graphObj);
```

Written by Kenneth S. Paulsen

► delete ↑

```
obj = delete(obj,graphObj)
```

Description:

Delete specified graph object if found.

Caution : When an nb_graph_ts or nb_graph_cs object is added to an nb_graph_subplot object, it will be copied. So if you provided the same object as you add, it will not be found and deleted.

Input:

- obj : An object of class nb_graph_subplot
- graphObj : An object of class nb_graph_ts or nb_graph_cs.

Example:

```
obj.delete(graphObj);
```

Written by Kenneth S. Paulsen

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_graph_subplot

Input:

- obj : An object of class nb_graph_subplot
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

See also:

Written by Kenneth S. Paulsen

► **getCode** ↑

```
[code,numOfGraphs] = getCode(obj,frameTitle,frameSubTitle,...  
                               footer,source,pageNumber,theme)
```

Description:

Get the latex code when including a graph in a slide object produced by an object of class nb_graph_subplot.

This method is called from the nb_Slide class, when making a beamer slide in latex.

Input:

- obj : An object of class nb_graph_subplot
- frameTitle : A string with the frame title of the slide.

- frameSubTitle : A string with the frame subtitle of the slide.
- footer : A cellstr with the footers of the slide
- source : A cellstr with the sources of the slide
- pageNumber : Give 1 if the page number should be included.
- theme : The beamer theme used. Default is '' (use default beamer theme).

Output:

- code : The latex code as a char.
- numOfGraphs : The number of graphs produced by the input object of class nb_graph_subplot. As an integer.

Examples:

Written by Kenneth S. Paulsen

► **graph ↑**

`graph(obj)`

Description:

Create a graph or more graphs

Graphs all the objects given to the nb_graph_subplot object.

Input:

- obj : An object of class nb_graph_subplot

Output:

The graph(s) plotted on the screen.

Examples:

`obj.graph();`

See also:

[nb_graph_subplot](#)

Written by Kenneth S. Paulsen

► **graphEng** ↑

graphEng(obj)

Description:

Plot the graph (english version)

Input:

- obj : An object of class nb_graph_adv

Output:

The graph plotted on the screen (english version)

Examples:

obj.graphEng()

Written by Kenneth S. Paulsen

► **graphNor** ↑

graphNor(obj)

Description:

Plot the graph (norwegian)

Input:

- obj : An object of class nb_graph_adv

Output:

The graph plotted on the screen (norwegian version)

Examples:

obj.graphNor()

Written by Kenneth S. Paulsen

► **remove** ↑

```
remove(obj,graphToRemove)
```

Description:

Remove specified graph object if found. Will give an error if not found.

Caution : When a nb_graph_obj or nb_graph_adv object is added to an nb_graph_subplot object, it will be copied. So if you provided the same object as you added (as the graphToRemove), it will not be found and a error will occure.

Input:

- obj : An object of class nb_graph_subplot
- graphToRemove : An object of class nb_graph_obj or nb_graph_adv.

Written by Kenneth S. Paulsen

► **reset** ↑

```
copiedGraph = reset(obj,oldGraphObj,newGraphObj)
```

Description:

Reset specified graph object if found, with the newGraphObj. Will give an error if not found.

Caution : If the resetting succeds the newGraphObj will be copied.

Caution : When an nb_graph_ts or nb_graph_cs object is added to an nb_graph_subplot object, it will be copied. So if you provided the same object as you add (as the oldGraphObj), it will not be found and a error will occure.

Input:

- obj : An object of class nb_graph_subplot
- oldGraphObj : An object of class nb_graph.
- newGraphObj : An object of class nb_graph.

Output:

- copiedGraph : The copied graph as an nb_graph object.

Written by Kenneth S. Paulsen

► **saveData** ↑

```
obj = saveData(obj,filename,strip,sheetName)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_graph_subplot
- filename : A string with the saved output name
- strip : > 'on' : Strip all observation dates where all the variables has no value. Default.
 > 'off' : Does not strip all observation dates where all the variables has no value.
- sheetName : A string with the name of the saved worksheets. I.e. the data of the subplots will be saved in the sheets with name [sheetName '<jj>']. E.g. if sheetName is given as 'GDP' the worksheet for the first subplot will get the name 'GDP(1)', while the second will get the name 'GDP(2)'. Default is 'Graph'.

Example:

```
obj.saveData('test');  
obj.saveData('test','on');  
obj.saveData('test','on','Name of figure');
```

Written by Kenneth S. Paulsen

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph_subplot object to (a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_graph_subplot
- extraName : If you want to add some extra name to the saveName property of the nb_graph_ts object before saving

the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)

- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Examples:

```
saveFigure(obj,'test','pdf');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Set properties of the nb_graph_subplot class.

See documentation NB Toolbox or type help('nb_graph_subplot') for more on the properties of the class.

Input:

- obj : An object of class nb_graph_subplot
- varargin : 'propertyName',PropertyValue,...

Output:

The input objects properties set to their new value.

Examples:

```
obj.set('subPlotSize',[2,2]);
```

Written by Kenneth S. Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the figure(s)

Input:

- obj : An object of class nb_graph_subplot

Output:

- obj : An object of class nb_graph_subplot, where the data is updated for the different graph objects.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ nb_graph_ts

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graph_ts(varargin)
```

Superclasses:

handle, nb_graph

Description:

This is a class for making timeseries graphics.

Constructor:

```
obj = nb_graph_ts(data)
```

Input:

- data : The input must one of the following:

> An object of class nb_ts. E.g. nb_graph_ts(data)

> A excel spreadsheet which could be read by the nb_ts class. E.g. nb_graph_ts('excelName1')

It is also possible to initialize an empty nb_graph_ts object. (Do not provide any inputs to the constructor)

Output:

- obj : An object of class nb_graph_ts

See also:

nb_graph, nb_graph_data, nb_graph_cs, nb_graph_adv, nb_graph_subplot

Written by Kenneth Sætherhagen Paulsen

Properties:

- DB
- a4Portrait
- addSpace
- annotation
- areaAccumulate
- averageLine
- axesFontSize
- axesFontWeight
- axesPrecision
- axesScaleLineWidth
- barAlpha2
- barLineWidth
- barShadingDate
- barWidth
- baseLineColor
- baseLineWidth
- candleIndicatorColor
- candleMarker
- candleWidth
- colorMap
- colorOrderRight
- crop
- dateInterpreter
- datesToPlot
- displayValue
- excelFooterEng
- excelTitleEng
- GraphStruct
- addAdvanced
- alignAxes
- areaAbrupt
- areaAlpha
- axesFast
- axesFontSizeX
- axesLineWidth
- axesScaleFactor
- barAlpha1
- barBlend
- barShadingColor
- barShadingDirection
- baseLine
- baseLineStyle
- baseValue
- candleIndicatorLineStyle
- candleVariables
- code
- colorOrder
- colors
- dashedLine
- dates
- defaultFans
- endGraph
- excelFooterNor
- excelTitleNor

- factor
- fanAlpha
- fanDatasets
- fanGradedLineWidth
- fanLegend
- fanLegendLocation
- fanMethod
- fanVariable
- figureColor
- figurePosition
- fileFormat
- flip
- fontScale
- graphStyle
- gridLineStyle
- highlightColor
- horizontalLineColor
- horizontalLineWidth
- legAuto
- legColor
- legColumns
- legFontSize
- legLocation
- legReorder
- legendText
- lineStyles
- lineWidths
- fakeLegend
- fanColor
- fanFile
- fanGradedStyle
- fanLegendFontSize
- fanLegendText
- fanPercentiles
- fanVariables
- figureName
- figureTitle
- findAxisLimitMethod
- fontName
- fontUnits
- grid
- highlight
- horizontalLine
- horizontalLineStyle
- language
- legBox
- legColumnWidth
- legFontColor
- legInterpreter
- legPosition
- legSpace
- legends
- lineWidth
- localVariables

- lookUpMatrix
- markerSize
- missingValues
- nanBeforeDate
- noLabel
- noTickMarkLabels
- noTickMarkLabelsRight
- normalized
- page
- patch
- pdfBook
- plotType
- position
- scatterDates
- scatterLineStyle
- scatterVariablesRight
- simpleRules
- spacing
- stopStrip
- subPlotSize
- subPlotsOptions
- tag
- titleAlignment
- titleFontWeight
- titlePlacement
- variablesToPlot
- verticalLine
- mYLim
- markers
- nanAfterDate
- nanVariables
- noLegend
- noTickMarkLabelsLeft
- noTitle
- numberOfGraphs
- parameters
- patchAlpha
- plotAspectRatio
- plotTypes
- saveName
- scatterDatesRight
- scatterVariables
- shading
- simpleRulesSecondData
- startGraph
- stopUpdate
- subPlotSpecial
- sumTo
- title
- titleFontSize
- titleInterpreter
- userData
- variablesToPlotRight
- verticalLineColor

- verticalLineLimit
- verticalLineWidth
- xLabelAlignment
- xLabelFontWeight
- xLabelPlacement
- xScale
- xTickLabelAlignment
- xTickLabels
- xTickRotation
- yDir
- yLabel
- yLabelFontWeight
- yLabelRight
- yLimRight
- yScale
- ySpacing
- verticalLineStyle
- xLabel
- xLabelFontSize
- xLabelInterpreter
- xLim
- xTickFrequency
- xTickLabelLocation
- xTickLocation
- xTickStart
- yDirRight
- yLabelFontSize
- yLabelInterpreter
- yLim
- yOffset
- yScaleRight
- ySpacingRight

- **DB** ↑

All the data given to the nb_graph_ts object collected in a nb_ts object. Should not be set using obj.DB = data! Instead see the nb_graph_ts.resetDataSource method.

- **GraphStruct** ↑

Sets the information on which variables and how to plot them when using the graphInfoStruct(...) method of this class.

Must be an .m-file with the code on how the structure of graphing information should be initialized or as a structure with the graphing information.

Only the graphInfoStruct(...) method uses this property. See the documentation of the NB toolbox for more on this input.

Inherited from superclass NB_GRAPH

- **a4Portrait** ↑

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass NB_GRAPH

- **addAdvanced** ↑

Set to false to prevent adding advanced components.

Inherited from superclass NB_GRAPH

- **addSpace** ↑

Sets the space before and/or after in the x direction of the graph [addedSpaceBefore addedSpaceAfter]. E.g. add one period before and after [1,1].

Inherited from superclass NB_GRAPH

- **alignAxes** ↑

Align axes at a given base value. Must be set to a scalar double. E.g. 0.

Inherited from superclass NB_GRAPH

- **annotation** ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the nb_annotation class for more. If you give more objects they must be collected in a cell array.

Examples can be found here:

\NBTOOLBOX\Examples\Graphics\nb_annotationExamples.m

Inherited from superclass NB_GRAPH

- **areaAbrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan. Default is false.

Inherited from superclass NB_GRAPH

- **areaAccumulate** ↑

Set if the areas should be accumulated or not. Default is true.

Inherited from superclass NB_GRAPH

- **areaAlpha** ↑

Sets the transparency of the area chart. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **averageLine** ↑

Set this property if you want to plot some average line(s).

Must be given as a cell, i.e. {var, settings, ...}:

> var : A string with the variable name you want to take the average of.

```

> settings : {'propertyName',PropertyValue,...}

The property names options are:

> 'calcStart' : The start date for when to start the
    calculation of the mean. Default is the
    date given by the 'startGraph' property.

> 'calcEnd'   : The end date for when to end the
    calculation of the mean. Default is the
    date given by the 'endGraph' property.

> 'color'     : Color of the average line. Default is
    [0 0 0] (MATLAB black).

> 'lineStyle' : Line style of the average line. Default is
    line style '-'. (Normal line)

> 'lineWidth' : Line width of the average line. Default
    line width is 2.5.

> 'plotStart' : The start date for when to start the
    plotting of the mean. Default is the
    date given by the 'startGraph' property.

> 'plotEnd'   : The end date for when to end the
    plotting of the mean. Default is the
    date given by the 'endGraph' property.

> 'side'       : The axes side. Either 'left' or 'right'.

If the variable is not found a proper error message will
occur.

```

- **axesFast** ↑

Set the if you want graphs to be produced fast with a loss of functionality and look or not. When empty default for the graph method is false, while for the graphSubPlots and graphInfoStruct methods are true.

Inherited from superclass NB_GRAPH

- **axesFontSize** ↑

Sets the font size of the axes tick mark labels, default is 12. Must be a scalar.

Inherited from superclass NB_GRAPH

- **axesFontSizeX** ↑

Sets the font size of the x-axis tick mark labels, default is []. I.e. use the axesFontSize property instead. Must be a scalar.

This property will not be scaled when fontUnits are changed, as is the case for axesFontSize.

Caution: Will not set the x-axis font size when plotType is set to scatter.

Inherited from superclass NB_GRAPH

- **axesFontWeight** ↑

Sets the font weight of the axes tick mark labels. Must be a string. Default is 'normal'. Either 'normal', 'bold', 'light' or 'demi'

Inherited from superclass NB_GRAPH

- **axesLineWidth** ↑

Sets the the line width of the axes, default is 0.5. Must be a set to a scalar double greater than 0.

Inherited from superclass NB_GRAPH

- **axesPrecision** ↑

Sets the precision/format of the rounding of number on the axes.

See the precision input to the nb_num2str function. Default is [], i.e. to call num2str without additional inputs.

Inherited from superclass NB_GRAPH

- **axesScaleFactor** ↑

Set the scaling factor used when axesScaleLineWidth is set to true.

Inherited from superclass NB_GRAPH

- **axesScaleLineWidth** ↑

Scale line width to axes height. true or false (default). If axesScaleFactor is set to a scalar number this will be the scaling factor used instead of the automatic scaling factor.

Inherited from superclass NB_GRAPH

- **barAlpha1** ↑

Set the alpha blending parameter 1, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barAlpha2** ↑

Set the alpha blending parameter 2, when barBlend is set to true. See nb_alpha. Default is 0.5

Inherited from superclass NB_GRAPH

- **barBlend** ↑

Set to true to do alpha blending instead of shading if shading options is used. Default is false.

Inherited from superclass NB_GRAPH

- **barLineWidth** ↑

Set the bar line width without affecting any other line widths. Default is empty, i.e. use the lineWidth property.

Inherited from superclass NB_GRAPH

- **barShadingColor** ↑

The color shaded bars are interpolated with. Default is 'white'. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **barShadingDate** ↑

Sets the date from which the bar plot should be shaded. Either a string or an object which is of a subclass of the nb_date class.

- **barShadingDirection** ↑

Sets the angle the shading should be vertical of. Either {'north'} | 'south' | 'west' | 'east'. As a string. Default is 'north'.

Inherited from superclass NB_GRAPH

- **barWidth** ↑

Sets the width of the bar plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **baseLine** ↑

If 0 is given the base line will not be plotted, otherwise it will be plotted.

Inherited from superclass NB_GRAPH

- **baseLineColor** ↑

Sets the color of the base line. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **baseLineStyle** ↑

Sets the line style of the base line. As a string. Either {'-' } | '--' | '---' | ':' | '-.' | 'none'.

Inherited from superclass NB_GRAPH

- **baseLineWidth** ↑

Sets the width of the base line. Must be a scalar.

Inherited from superclass NB_GRAPH

- **baseValue** ↑

Sets the base value used for bar and area plot. (Sest also the y-axis base value for the base line)

Inherited from superclass NB_GRAPH

- **candleIndicatorColor** ↑

Sets the color of the indicator of the candle. Must either be a 1 x 3 double with the RGB colors or a string with the color name.

Inherited from superclass NB_GRAPH

- **candleIndicatorLineStyle** ↑

Sets the line style of the indicator of the candle. Must be a string. Default is '-'.

Inherited from superclass NB_GRAPH

- **candleMarker** ↑

Sets the marker of the indicator of the candle. Must be a string. Default is 'none'.

Inherited from superclass NB_GRAPH

- **candleVariables** ↑

Sets which variables are going to be plotted as open, close, high, low and indicator. Must be a cellstr. E.g.

{'open','var1','close','var2',...}

If a type (e.g. 'low') is not given it will not be plotted.

Meaning of each type:

- > 'close' : The lowest values of the plotted patches of the candle.
- > 'high' : The highest values of the plotted candles.
- > 'indicator' : The value for where to plot a horizontal line (across the patch). E.g. the mean value.
- > 'open' : The highest values of the plotted patches of the candle.
- > 'low' : The lowest values of the plotted candles.

Inherited from superclass NB_GRAPH

- **candleWidth** ↑

Sets the width of the candle plot. As a scalar. Default is 0.45.

Inherited from superclass NB_GRAPH

- **code** ↑

If you set this to a file name, this file will be evaluated after all the plotting is done. Must be a string.

This could be nice for adding special annotation, lines and so on. In this file you can use any MATLAB function, but some strange error message can occur if you use local variables which is already defined (So use long and descriptive variable names in the provided code to prevent this).

This property is only an option for the method graph.

Nice to know:

> The current nb_figure object can be reached with
get(obj,'figureHandle')

> The current nb_axes object can be reached with
get(obj,'axesHandle')

Caution : Be aware that the name of the file must be unique and cannot be a MATLAB function or a local variable.

Caution : The file must be a MATLAB .m file

Inherited from superclass NB_GRAPH

- **colorMap** ↑

Sets the colormap used when plotType is set to 'image' and fanMethod is set to 'graded'. Must be given as n x 3 double or the path to a .mat file that contain the colormap. Default is given by nb_axes.defaultColorMap.

For an example of a supported MAT file see:

- ...\\Examples\\Graphics\\colorMapNB.mat

Inherited from superclass NB_GRAPH

- **colorOrder** ↑

Sets the color order of the plotted data (Left axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the left axes.

Inherited from superclass NB_GRAPH

- **colorOrderRight** ↑

Sets the color order of the plotted data (Right axes). Either a cellstr with the color names (size; 1xM) or a double with RGB colors (Size; Mx3). Where M is the number of plotted variables against the right axes.

Inherited from superclass NB_GRAPH

- **colors** ↑

Sets the colors of the given variable(s). Must be a cell array on the form: {'var1', 'black', 'var2', [0.2 0.2 0.2],...}. Here we set the colors of the variables 'var1' and 'var2' to 'black' and [0.2 0.2 0.2] (RGB color) respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Caution: For the graphInfoStruct() method you need to select the variables from obj.DB.dataNames instead!

Inherited from superclass NB_GRAPH

- **crop** ↑

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_GRAPH

- **dashedLine** ↑

Sets the date at which all the plotted lines will be dashed. Must be given as a string or an object which is of a subclass of the nb_date class.

- **dateInterpreter** ↑

Sets how to interpreter the dates. I.e. where to place the datapoints in relation to the tick mark labels. Must be a string. {'start'} | 'end' | 'middle'.

- **dates** ↑

All the dates of the plot as a cellstr. Not settable.

- **datesToPlot** ↑

Dates to plot as a cellstr of dates (strings). Can also use local variables.

This property can be used if you want to compare time observations against eachother. See nb_graph_tsExamples.m for more.

- **defaultFans** ↑

If a date is given default fans for the one of the variables 'QSA_GAPNB_Y', 'QSA_GAP_Y', 'QUA_DPY_PCPI', 'QUA_DPY_PCPIJAE', 'QUA_DPY_PCPIXE', 'QUA_DPY_PCPIXE_R' and 'QUA_RNFOOLIO' is added, otherwise not. The frequency of the provided data must be quarterly (4). Either as a string or an nb_quarter object. Only one of the aforementioned variables should be plotted at a time.

- **displayValue** [↑](#)

Set to false to not display observation values when holding the mouse over the graph. Default is true.

Inherited from superclass NB_GRAPH

- **endGraph** [↑](#)

Sets the end date of the graph. Either as string or an object which is of a subclass of the nb_date class. If the given date is after the end date of the data, the data will be appended with nan values. I.e. the graph will be blank for these dates.

- **excelFooterEng** [↑](#)

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. English version.

Inherited from superclass NB_GRAPH

- **excelFooterNor** [↑](#)

If you want a extended footer to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to include the same footer as in the graph. Norwegian version.

Inherited from superclass NB_GRAPH

- **excelTitleEng** [↑](#)

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). English version.

Inherited from superclass NB_GRAPH

- **excelTitleNor** [↑](#)

If you want a custom title to the graph in the excel spreadsheet produced by the nb_graph_package.saveData method, this property can be used. The default is to use the same title as in the graph (panel if multiple graphs). Norwegian version.

Inherited from superclass NB_GRAPH

- **factor** [↑](#)

Set this property to multiply the data with a given factor.
Must be a scalar.

Inherited from superclass NB_GRAPH

- **fakeLegend** ↑

Set this property if you want to add a legend of none plotted variable. You can set this property. Must be a cell on the form: {'legendName', settings,...}, where the legendName will be text of the fake legend. And the settings input must also be a cell array with the optional inputs {...,'propertyName', propertyName,...}. These settings will set the appearance of the fake legend. The following propertyNames are supported:

```
> 'type'      : Either 'line' or 'patch'.  
  
> 'color'     : Sets the color of the fake legend. Default is  
    'black'. Must either be a 1 x 3 double with  
    the RGB colors or a string with the color  
    name. If shaded patch is wanted it must be  
    2 x 3 double with the RGB colors or a 1 x 2  
    cellstr with the color names to use.  
  
> 'direction' : Sets the direction of the shading. Either  
    {'north'} | 'south' | 'west' | 'east'. As a  
    string. Default is 'north'. Only an option  
    when 'type' is set to 'patch'.  
  
> 'edgeColor'  : Sets the color of the edge of the patch.  
    Default is the same as the 'color' option.  
    Only an option when 'type' is set to 'patch'.  
  
> 'lineStyle'   : Sets the line style of the fake legend. As a  
    string. Either {'-' } | '--' | '---' | ':' |  
    '-.' | 'none'. Default is a normal line.  
  
    Will set the edge line style if the 'type'  
    is set to 'patch'.  
  
> 'lineWidth'   : Sets the line width of the fake legend.  
    Default is 2.5. Must be a scalar.  
  
> 'marker'     : Sets the marker of the fake legend. Default is  
    'none'. Only an option when 'type' is set to  
    'line'. Lookup LineSpec in the MATLAB help  
    meny for more on the supported marker types.  
  
Caution : Even if you don't want to set any of the optional  
    input. The settings input must be given. I.e an  
    empty cell, i.e. {}.
```

Inherited from superclass NB_GRAPH

- **fanAlpha** ↑

Sets the transparency of the fan chart. May be needed if you add fan charts to more than one variable at the time. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

- **fanColor** ↑

Sets the colors of the fan charts. Must be a matrix of size; number of fan layer x 3, with the RGB color specifications.

or

A string with the basis color for the fan colors. Either 'red', 'blue', 'green', 'yellow', 'magenta', 'cyan' or 'white'.

Be aware that the classification of the color of the given "basecolor" is based on the property fanPercentiles, so if you have more or less than 4 layers you must change this property to have the same size as the number of layers. (This is the case even if you don't use it to calculate the percentiles of a given datasets. Given by the property fanDatasets)

Default is the nb colors.

Caution: If fanMethod is set to 'graded' the color used are those assign to the colorMap property.

- **fanDatasets** ↑

Sets the data of the added fan layers.

If you have the data of the different layer in separate datasets. Then the input to this property must be a 2 * number of layers cell matrix. Can be excel spreadsheets, .mat files or dyn_ts objects.

or

You can also give the data with all the simulated data or fan layers in an dyn_ts object or an nb_ts object. (Each page one simulation or each page one layer). (FASTEST with an nb_ts object as input.)

Remember that the percentiles for the fan charts is set by the property fanPercentiles, default is [.3, .5, .7, .9].

The fan layers are added differently given the method you use.

> graph(...): Adds the fan given in this property to the last variable given in the property variablesToPlot. (If not the fanVariable property is set.) If you graph more datasets (in separate figures), the fan layers will be the same for all of them. Set the fanVariables property if that is not wanted.

> graphSubPlots(...): Adds the fan given in this property to the corresponding variable in the last given datasets (last page of the nb_ts object given by the property DB) of each each subplot.

-graphInfoStruct(...): Adds the fan given by this property to each variable of each subplot. Added for the last given dataset. (Last page of the nb_ts object given by the property DB.)

- **fanFile** ↑

The file to get the default fan layers from. As a string.

- **fanGradedLineWidth** ↑

Line width used but the graded fan chart is set to

- **fanGradedStyle** ↑

Line width used but the graded fan chart is set to

- **fanLegend** ↑

Set this property if you want to add a MPR styled legend of the fan layers. Set it to 1 if wanted. Default is not (0).

- **fanLegendFontSize** ↑

Sets the font size of the fan legend. Default is 12. Must be a scalar.

- **fanLegendLocation** ↑

Sets the location of the fan legend. Locations are:

```
> 'Best', (same as 'North'.)
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'outside' : Places the legend outside the axes, one the right side. (If you create subplots, this is the only option which will look good.)
```

You can also give a 1 x 2 double vector with the location of where to place the legend. First column is the x-axis location and the second column is the y-axis location. Both must be between 0 and 1.

- **fanLegendText** ↑

Set this property if you want to give the text over the colored boxes of the MPR looking fan legend, you must give them as a cell. (Must have same size as number of layers.) Default is the percentiles with the % sign added. E.g. {'30%','50%','70%','90%'}.

- **fanMethod** ↑

{'percentiles'} | 'hdi' | 'graded'; Which type of fan chart is going to be made. 'percentiles' are produced by using percentiles, 'hdi' use highest density intervals, while 'graded' creates a graded fan chart of the colors specified with the colorMap property.

- **fanPercentiles** ↑

Sets the percentiles you want to calculate (if the fanDatasets property is representing simulation) or specify the percentiles you have given (if the fanDatasets property is representing already calculated percentiles). Must be given as a double vector. Default is [0.3 0.5 0.7 0.9].

This property is also the default base for the text of the fan legend. (Where the percentiles are given in percent.)

- **fanVariable** ↑

Sets the variable you want to add the fan layers to. Must be given as a string with the variable name. Default is to use the last variable in the variablesToPlot property.

If this input is a cellstr with more than one variable, the 'fanPercentiles' input must be a scalar double. Fan chart will then be added to more than one variable.

- **fanVariables** ↑

If you have the fan layers included in the same dataset as your other plotted variables you can use this property. The input must then be given as 1 x 2 * number of layers cell array with the fan layers names (as strings). I.e. both the upper and lower layers must be given as separate variables.
(Need not be ordered though.)

E.g. {'LowPerc 90%', 'LowPerc 70%', 'UppPerc 90%', 'UppPerc 70%'}

Caution: Must fit the number of percentiles given by the property fanPercentiles (Remember to include both upper and lower bounds!).

Only an input to the method graph(...): Adds the given fan layers to the last variable given by the property variablesToPlot or the variable given by fanVariable property.

- **figureColor** ↑

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_GRAPH

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_GRAPH

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_GRAPH

- **figureTitle** ↑

Sets if figure titles is wanted on the figures when using the method graphInfoStruct. The fieldnames of the GraphStruct property will be used as figure title. Default is 0 (not include).

If set to a one line char this property can be used for the plotting method graphSubPlots.

Inherited from superclass NB_GRAPH

- **fileFormat** ↑

Sets the save file format. Either 'eps', 'jpg', 'pdf' or 'png'. 'pdf' is default.

Inherited from superclass NB_GRAPH

- **findAxisLimitMethod** ↑

Sets the method used for finding the axis limits, and y-ticks.

- > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
- > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
- > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
- > 4 : Uses the MATLAB algorithm for finding the axis limits.

Inherited from superclass NB_GRAPH

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_GRAPH

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_GRAPH

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_GRAPH

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : Don't use this property in combination with one of the graph styles defined by this class. I.e. 'mpr', 'mpr_white', 'presentation' and 'presentation_white' They will set this property to 'normalized'.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_GRAPH

- **graphStyle** ↑

Sets the graphing style of the plots. Give 'mpr' if you want the MPR looking style, or 'presentation' for graphs for the presentation (see nb_Presentation). If you don't want the grey shaded background you can use 'mpr_white' and 'presentation_white' instead.

You can also add your own graph style setting through a .m file. I.e. give the filename (without extension) as a string.

E.g. In the file you can type in something like:

```
set(obj,'titleFontSize',12,'legInterpreter','tex');
```

Caution: If you give this as the first input to the set method it is possible to overrun some of the default settings if that is wanted. E.g. font size of the text, the shading of the plot and so on.

Inherited from superclass NB_GRAPH

- **grid** ↑

Set it to 'on' if grid lines are wanted otherwise set it to 'off'. Default is 'off'.

Inherited from superclass NB_GRAPH

- **gridLineStyle** ↑

Sets the line style of the grid lines. Either '-' , '--' , ':' or '-.' .

Inherited from superclass NB_GRAPH

- **highlight** [↑](#)

If highlighted area between two dates of the graph is wanted set this property.

One highlighted area:

A cell array on the form {'date1', 'date2'}

More highlighted areas:

A nested cell array on the form

{{'date1(1)', 'date2(1)'}, {'date1(2)', 'date2(2)'}, ...}

If plotType is set to 'scatter' scalars can be used instead.

- **highlightColor** [↑](#)

Sets the colors of the background patches. Must be one of the following:

- > A string with the color name to use when plotting (all) the patch(es)
- > A cell array of strings with the color names
- > A cell with 1 x 3 doubles with the RGB color specification.

When given as a cell this property must match the highlight property. I.e. the number of highlighted areas added.

The default color of the highlighted areas is 'light blue'.

- **horizontalLine** [↑](#)

If you want to include some extra horizontal lines, beside the base line, you can set this property. Must be set to a scalar or a double vector with the y-axis value(s) on where to place the horizontal line(s).

Inherited from superclass NB_GRAPH

- **horizontalLineColor** [↑](#)

Sets the color(s) of the given horizontal line(s).

Must either be an M x 3 double with the RGB color(s) or a 1 x M cell array of strings with the color name(s). Where M must be less than the number of plotted horizontal lines.

If less or no color(s) is/are set by this property the default colors for the rest or all the horizontal line(s) is/are [0 0 0], i.e. MATLAB black.

Inherited from superclass NB_GRAPH

- **horizontalLineStyle** [↑](#)

Sets the style(s) of the given horizontal line(s).

Must either be a string or a $1 \times M$ cell array of strings with the style(s). Where M must be less than the number of plotted horizontal lines.

If less or no style(s) is/are set by this property the default lines for the rest or all the horizontal line(s) is/are '-'.

Inherited from superclass NB_GRAPH

- **horizontalLineWidth** [↑](#)

Sets the line width of the horizontal line(s). Must be a scalar. Default is 1.

Inherited from superclass NB_GRAPH

- **language** [↑](#)

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_GRAPH

- **legAuto** [↑](#)

'on' | {'off'}. As a string.

> 'on' : The strings given through the fakeLegend and patch properties are automatically added to the legend.

> 'off' : The strings given through the fakeLegend and patch properties are not automatically added to the legend. Which means that you must provide all the legend information through the legends property. The patch descriptions must be given first and the fake legend description must be given last in the legends property. Default.

Inherited from superclass NB_GRAPH

- **legBox** [↑](#)

Set if a box should be drawn around the legend. Either {'on'} | 'off'.

Caution : When removing the box you make it impossible to move it around afterwards.

Inherited from superclass NB_GRAPH

- **legColor** [↑](#)

Sets the color of the background of the box of the legend. Either as 1×3 double with the RGB or as a string). Default is 'none', i.e. transparent.

Inherited from superclass NB_GRAPH

- **legColumnWidth** ↑

Sets the width of the columns of the legend. Default is to let MATLAB find the width itself. If given it must be a scalar with the width of all columns of the legend or a $1 \times \text{legColumns}$ double with the width of each individual column.

Should be between 0 and 1. Use the try and fail method to find out what fits.

Inherited from superclass NB_GRAPH

- **legColumns** ↑

Sets the number of columns of the legends. Must be a scalar.

Inherited from superclass NB_GRAPH

- **legFontColor** ↑

Sets the font color(s) of the legend text. Must either be a 1×3 double or a string with the color name of all legend text objects, or a $M \times 3$ double or a cellstr array with size $1 \times M$ with color names to use of each legend text object.

Inherited from superclass NB_GRAPH

- **legFontSize** ↑

Sets the font size of the legend. Must be a scalar default is 10.

Inherited from superclass NB_GRAPH

- **legInterpreter** ↑

Sets the interpreter of the legend text. Must be a string.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **legLocation** ↑

Sets the location of the legend, must be string. The location can be:

> 'Best' : Same as 'NorthWest'.
> 'NorthWest'
> 'North'
> 'NorthEast'
> 'SouthEast'
> 'South'
> 'SouthWest'
> 'below' : Below the plot (when many subplots, it will place the legend below all of them)
> 'middle' : Could only be use for 2×2 subplot, and then it places the legend between the upper and lower subplots.

Location that only works for the graph() method:

- > 'outsideright' : Legend is placed outside the axes on the right side, and the axes is resized, so the axes and legend does not take up more space than the axes did in the first place. The legend is vertically centered.
- > 'outsiderighttop' : Same as 'outsideright', but placed vertically top.

Inherited from superclass NB_GRAPH

- **legPosition** ↑

Sets the position of the legend. Must be a 1x2 double. Where the first element is the x-axis location and the second is the y-axis location. [xPosition yPosition]

Caution : Both elements must be between 0 and 1. And where we have that [0, 0] will be the bottom left location of the axes and [1, 1] is the top right position of the axes. (This position will be the top left position of the legend itself.)

Caution : This option overruns the legLocation property.

Inherited from superclass NB_GRAPH

- **legReorder** ↑

Set this property to reorder the legend. Either a string with {'default'} | 'inverse' or a double with how to reorder the legend. E.g. if you have 3 legends to plot the default index will be [1,2,3], the inverse ('inv') will be [3,2,1]. If you want to decide that the 3. legend should be first, then the 1. and 2., you can type in [3,1,2].

Caution : If a double is given it must have as many elements as there is plotted legends. I.e. if you give a empty string to one of the legends it will be excluded. Say you provide the property legends as {'s','','t','f'} then the double must have size 1x3.

Inherited from superclass NB_GRAPH

- **legSpace** ↑

The vertical space between the texts of the legend. Must be a scalar. Default is 0.

Inherited from superclass NB_GRAPH

- **legendText** ↑

Sets the legend(s) of the given variable(s). Must be a cell array on the form: {'var1', 'Name', 'var2', 'Name2', ...}. Here we set the legends of the variables 'var1' and 'var2' to 'Name' and 'Name2' respectively. (The rest will be given the variable names as default).

```

To give a legend to a splitted line (second part) give;
{...,'Var1(second)','LegendSplitted',...} (Is case sensitiv)

Caution : legAuto should be set to 'on'. Which is not default.
          This is only important if the patch or fakeLegend
          property is used.

Caution : When given this property will overwrite the legends
          property

Caution : If the variable, scattergroup or candle is not found
          no error will occure!

Caution : If you have used the patch or fakeLegend property these
          can be translated as well with this property

Inherited from superclass NB_GRAPH

- legends ↑

Give the legends of the plot. Must be given as a cell array
of strings.

Default (it depends on the method you use):

> 'graph'           : The variable names are used as the
                      default legends. (Given by the
                      variableToPlot and variableToPlotRight
                      properties)

> 'graphSubPlot'   : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)

> 'graphInfoStruct' : The dataset names are used as the
                      default legends. (Given by the
                      dataNames of the DB property.)

Caution : The ordering of the legends is as follows:

- patch             : See the patch property (only when
                      legAuto is set to 'off')

- variablesToPlot   : Then the all the variablesToPlot
                      variables. When You have splitted
                      lines, i.e. either using the
                      dashedLine property or the
                      lineStyles property, these can be set
                      after all the variablesToPlot. Have
                      in mind that the ordering is kept.

- variablesToPlotRight : Then the comes all the variables
                        provided by the variablesToPlotRight.
                        Then the splitted lines.

- fakeLegend        : Then you can provide the legend text
                      for the fake legend. (only when
                      legAuto is set to 'off')

```

Example:

```
data      = nb_ts.rand('2012',10,2);
plotter = nb_graph_ts(data);
plotter.set('variablesToPlot',{'Var1'},...
            'variablesToPlotRight',{'Var2'},...
            'lineStyles',{'Var1',{'-','2014','--'},...
                         'Var2',{'-','2014','--'},...
            'legends',{'Test','Test (--)','Test2','Test2 (--)'});
plotter.graph()
```

Inherited from superclass NB_GRAPH

- **lineStyles** ↑

Sets the line styles of the given variable(s). Must be a cell array on the form: {'var1', '-' , 'var2', '--', ...}. Here we set the line styles of the variables 'var1' and 'var2' to '-' and '--' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Supported line styles are '-','--','::','.-','---

Caution : For nb_graph_ts and nb_graph_data it is possible to split the plotted line using the following syntax;
{'var1',{'-','2000Q1','--'}} and {'var1',{'-',2,'--'}} respectively

Inherited from superclass NB_GRAPH

- **lineWidth** ↑

Sets the line width of the line plots. Must be a scalar.
Default is 2.5.

Inherited from superclass NB_GRAPH

- **lineWidths** ↑

Sets the line widths of the given variables. Must be a cell array on the form: {'var1', 1, 'var2', 1.5,...}. Here we set the line widths of the variables 'var1' and 'var2' to 1 and 1.5 respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

Inherited from superclass NB_GRAPH

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_GRAPH

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
```

```
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {  
'mnemonics1','englishDescription1','norwegianDescription1';  
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_GRAPH

- **mYLim** ↑

Sets the max/min y-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty this setting only sets the left y-axis limits.

Only an option for the graphInfoStruct(...) method.

- **markerSize** ↑

Sets the size of the all markers. Must be a scalar. Default is 8.

Inherited from superclass NB_GRAPH

- **markers** ↑

Sets the markers of the given variables. Must be a cell array on the form: {'var1', '^', 'var2', 'o', ...}. Here we set the markers of the variables 'var1' and 'var2' to '^' and 'o' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the table below on which markers that are supported.

'+'	: Plus sign
'o'	: Circle
'*'	: Asterisk
'.'	: Point
'x'	: Cross
's'	: Square
'd'	: Diamond
'^'	: Upward-pointing triangle
'v'	: Downward-pointing triangle
'>'	: Right-pointing triangle
'<'	: Left-pointing triangle
'p'	: Five-pointed star (pentagram)
'h'	: Six-pointed star (hexagram)

Inherited from superclass NB_GRAPH

- **missingValues** ↑

Set how to interpret missing observations. Must be a string.
 Either;

```
> 'interpolate' : Linearly interpolate the missing values.

> 'strip'      : Strip the missing values. Should only be
                  used with business days data.

> 'both'       : First strip up until a date given by the
                  property stopStrip and then interpolate the
                  rest of the missing values. If the stopStrip
                  property is not given this will result in
                  the same as the 'strip' option.

%
> 'none'        : No interpolation (default)
```

- **nanAfterDate** [↑](#)

If setted this property would make the graph blank after the provided date, but the x-axis would still keeps its axis limits. Must be a string with the date or an object which of a subclass of the nb_date class.

- **nanBeforeDate** [↑](#)

If setted this property would make the graph blank before the provided date, but the x-axis would still keeps its axis limits. Must be a string with the date or an object which of a subclass of the nb_date class.

- **nanVariables** [↑](#)

Sets the nan option for individual variables. Must be given as a cell. I.e. {'var1', {'2012Q2', '2012Q4'}, ... 'var2', {'2012Q2', '2012Q4'}, ...}. This will make both 'var1' and 'var2' variables blank before '2012Q2' and after '2012Q4' (Not including.) in the given graph.

- **noLabel** [↑](#)

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Both on the y-axis and x-axis. Default is 0. This is of course only have an effect if the nb_graph object have been given the xLabel or yLabel properties. (These are empty by default.)

Inherited from superclass NB_GRAPH

- **noLegend** [↑](#)

Set it to 1 if no legend is wanted, otherwise set it to 0. Default is 0.

Inherited from superclass NB_GRAPH

- **noTickMarkLabels** [↑](#)

Set to true (1) to remove tick marks and tick marks label of axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsLeft** ↑

Set to true (1) to remove tick marks and tick marks label of the left side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTickMarkLabelsRight** ↑

Set to true (1) to remove tick marks and tick marks label of the right side of the axes. Default is false (0).

Inherited from superclass NB_GRAPH

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the graphs, otherwise set it to 0. Default is 0.

Set it to 2 if you want the dataNames{ii} as the title of the graph when using the graph method.

Inherited from superclass NB_GRAPH

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_GRAPH

- **numberOfGraphs** ↑

Number of graph produced by the graph methods. Not settable.

Inherited from superclass NB_GRAPH

- **page** ↑

Sets the page of the data object to plot. Only an option for graph() method. Default is [], i.e. plot all pages in separate figures.

Inherited from superclass NB_GRAPH

- **parameters** ↑

A struct with the parameters that can be used in evaluation of expressions in the graphSubPlots and graphInfoStruct methods.

Inherited from superclass NB_GRAPH

- **patch** ↑

Sets the patch property of the object. If set this will add a patch (color fill) between two variables.

Only one patch :

```
{'patchName','var1','var2',color}
```

More patches :

```
{'patchName1','var1Patch1','var2Patch1',color1 ,...  
'patchName2','var1Patch2','var2Patch2',color2,...}
```

Where the the color option(s) must either be a 1 x 3 double with the RGB colors or a string with the color name.

The first input will be the legend text of the patch. I.e. 'patchName', 'patchName1' and 'patchName2' will be the description text included in the legend. (If not the property legAuto is set to 'off'.)

Inherited from superclass NB_GRAPH

- **patchAlpha** [↑](#)

Sets the transparency of the patches. A number between 0 and 1. 1 is opaque, while 0 is fully transparent.

Inherited from superclass NB_GRAPH

- **pdfBook** [↑](#)

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_GRAPH

- **plotAspectRatio** [↑](#)

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_GRAPH

- **plotType** [↑](#)

Sets the plot type. Either:

> 'line' : If line plot(s) is wanted. Default.

> 'stacked' : If stacked bar plot(s) is wanted. Not an option for the graphSubPlots(...) method.

> 'grouped' : If grouped bar plot(s) is wanted.

> 'dec' : If decomposition plot(s) is wanted. Which is

a bar plot with also a line with the sum of the stacked bars. Not an option for the graphSubPlots(...) method.

```

> 'area'      : If area plot(s) is wanted.

> 'candle'    : If candle plot(s) is wanted.

> 'scatter'   : If scatter plot(s) is wanted.

> 'simpleRules' : Can be given if you want graph of the interest rate path compared with the simple rules. (MPR looking graph.) The different variables must have the names:

'QUA_RNFOLIO'      : Interest rate path.
'QUA_TAYLOR'        : Taylor-rule.
'QUA_GROWTHRULE'   : Uses a growth gap measure instead of the output gap in the simple rule.
'QUA_FOREIGNRULE'  : A simple rule that takes foreign variables into account
'QUA_MODELBASEDRULE': A simple rule base on some work of FA. A rule which is robust in different models.

```

You can set which simple rules you want to plot with the property simpleRules and you can graph two different versions of the simple rules using the simpleRulesSecondData property.

Only an option for the graph(...) method.

- **plotTypes** ↑

Sets the plot type of the given variables. Must be a cell array on the form: {'var1', 'line', 'var2', 'grouped', ...}. Here we set the plot type of the variables 'var1' and 'var2' to 'line' and 'grouped' respectively. The variables given must be included in the variablesToPlot or variablesToPlotRight properties.

See the property plotType above on which plot types that are supported.

- **position** ↑

Sets the position of the axes, must be an 1x4 double. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

For the graph method graphSubPlots the position input can be given a 1 x N cell array, where each element is a 1x4 double. N = obj.subPlotSize(1)*obj.subPlotSize(2).

Inherited from superclass NB_GRAPH

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

Inherited from superclass NB_GRAPH

- **scatterDates** ↑

Sets the start and end to be used for the scatter plot plotted against the left axes. Must be a nested cell array. The dates can be given as strings or date objects. E.g:

```
{'scatterGroup1',{'startDate1','endDate1'},...  
 'scatterGroup2',{'startDate2','endDate2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterDatesRight)

Be aware : Each date will result in one point in the scatter.

- **scatterDatesRight** ↑

Sets the start and end to be used for the scatter plot plotted against the right axes. Must be a nested cell array. The dates can be given as strings or date objects. E.g:

```
{'scatterGroup1',{'startDate1','endDate1'},...  
 'scatterGroup2',{'startDate2','endDate2'},...}
```

Caution : This property must be provided to produce a scatter plot! (Or of course scatterDates)

Be aware : Each date will result in one point in the scatter.

- **scatterLineStyle** ↑

Sets the scatter line style. Must be string. See 'lineStyles' for supported inputs. (Sets the line style of all scatter plots)

- **scatterVariables** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the left y-axis. E.g:

```
{'var1','var2'}
```

- **scatterVariablesRight** ↑

The variables used for the scatter plot. Must be a 1x2 cellstr array. The first element will be the variable plotted against the x-axis, while the second element will be the variable plotted against the right y-axis. E.g:

```
{'var1','var2'}
```

- **shading** ↑

The shading option of the axes background:

- 'grey' : Shaded grey background
- 'none' : Background color given by the color property.
- A n x m x 3 double with the background color. n is the number of horizontal pixels, m is the number of vertical pixels and 3 means the RGB colors.

Inherited from superclass NB_GRAPH

- **simpleRules** ↑

Sets the simple rules which will be graphed by the method graph(...) when the plotType property is set to 'simpleRules'. Must be a cell array of strings. See the documentation on the property plotType for more on the simple rules to include.

- **simpleRulesSecondData** ↑

Sets the data of an older version of the simple rules. Only an option of the method graph(...) when the plotType property is set to 'simpleRules'. Must be a string with the name of a excel spreadsheet, a dyn_ts object or an nb_ts object. (.mat file is not supported.) Must include the variables given below (or the variables given by the 'simpleRules' property.):

```
> 'QUA_TAYLOR'      : Taylor-rule.  
> 'QUA_GROWTHRULE'  : Uses a growth gap measure instead of  
                      the output gap in the simple rule.  
> 'QUA_FOREIGNRULE' : A simple rule that takes foreign  
                      variables into account.  
> 'QUA_MODELBASEDRULE' : A simple rule base on some work of  
                          FA. A rule which is robust in  
                          different models.
```

- **spacing** ↑

Sets the x-ticks spacing. Must be a scalar. Refers to the given x-axis tick frequency.

- **startGraph** ↑

Sets the start date of the graph. Must be a string or an object which of a subclass of the nb_date class. If this date is before the start date of the data, the data will be expanded with nan (blank values) for these periods.

- **stopStrip** ↑

Sets the stop date of the graph. Must be a string or an object which of a subclass of the nb_date class.

- **stopUpdate** ↑

Sets a time for when to stop updating the data of this graph, i.e. if the actual time has past this date the data will not be updated. Must be a string on the format 'yyyymmdd', 'yyyymmddhh' or 'yyyymmddhhmm'. E.g. '20121010', '2012101010' or '201210101000'.

Caution: No warning will be provided!

- **subPlotSize** ↑

Sets the number of subplots per figure. Must be a 1 x 2 double with the number of subplot rows as the first element and the number of subplot columns as the second element. Only an option of the graphSubPlots() and graphInfoStruct() methods.

Default is [2, 2].

Inherited from superclass NB_GRAPH

- **subPlotSpecial** ↑

Set it to 1 if you want the 1x2 and 2x1 subplots to better fit for usage in presentations. Default is 0, i.e. use MATLAB default positions.

When set to 1 this class uses the nb_subplotSpecial instead of nb_subplot to find the positions of the subplots.

Inherited from superclass NB_GRAPH

- **subPlotsOptions** ↑

Use this property to set subplot spesific options of each variable while using the the method graphSubPlots(). Each field must be the name of the variable. The options are:

- **yLabel** : The y-axis label.
- **yLim** : Sets the y-axis limits. As a 1x2 double.
- **ySpacing** : Spacing between y-axis tick marks.
- **dashedLine** : The date to start the dashed line for the given variable.

Example:

```
s.Var1 = struct('yLim',[0,1]);  
s.Var2 = struct('yLim',[0,1]);
```

- **sumTo** ↑

Sets a number which the area or the stacked bar plot should sum to. Must be a scalar. I.e. if 100 is given. The sizes of the bars will be the percentage share of the total sum.

Inherited from superclass NB_GRAPH

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_GRAPH

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_GRAPH

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_GRAPH

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_GRAPH

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_GRAPH

- **variablesToPlot** ↑

Sets the variables to plot against the left (or both) axes.
Must be a cell array of strings with the variable names. I.e.
{'var1', 'var2', ...}.

Not an option for the graphInfoStruct(...), which use the
property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **variablesToPlotRight** ↑

Sets the variables to plot against the right axes. Must be
a cell array of strings with the variable names. I.e.
{'var1', 'var2', ...}.

Not an option for the graphInfoStruct(...), which use the
property GraphStruct instead.

Inherited from superclass NB_GRAPH

- **verticalLine** ↑

Sets the date(s) of where to place a vertical line(s).
(Dotted orange line is default). Must be a string or a cell
array of strings with the date(s) for where to place the
vertical line(s). I.e. 'date1' or {'date1','date2',...}

Each element of the cell could also be given as a cell with
two dates, i.e. {'date1', 'date2' }. Then the vertical line
will be placed between the given dates.

If plotType is set to 'scatter' scalars can be used
instead.

- **verticalLineColor** ↑

Sets the color(s) of the given vertical line(s). Must either
be an M x 3 double with the RGB color(s) or a 1 x M cell array
of strings with the color name(s). Where M must be less than
the number of plotted vertical lines.

If less or no color(s) is/are set by this property the default
color(s) for the rest or all the vertical line(s) is/are
[0 0 0]. I.e. MATLAB black.

- **verticalLineLimit** ↑

Sets the y-axis limit(s) of the given vertical line(s). Must
either be a 1 x M cell array of 1 x 2 double with the upper
and lower y-axis limit(s). Where M must be less than the
number of plotted vertical lines.

If less or no limit(s) is/are set by this property the default
limit for the rest or all the vertical line(s) is/are the full
height of the graph.

- **verticalLineStyle** ↑

Sets the line style(s) of the given vertical line(s). Must either be a 1 x M cell array of strings with the style(s). Where M must be less than the number of plotted vertical lines.

If less or no style(s) is/are set by this property the default line style is/are used for the rest or all the vertical line(s) is/are '--'.

- **verticalLineWidth** ↑

Sets the line width of (all) the vertical line(s). Must be a scalar. Default is 1.5.

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_GRAPH

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar. Default is 12.

Inherited from superclass NB_GRAPH

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string. Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_GRAPH

- **xLim** ↑

Sets the x-axis limits of the plot. Must be 1 x 2 double vector, where the first number is the lower limit and the second number is the upper limit. (Both or one of them can be set to nan, i.e. the default limits will be used) E.g. [0 6], [nan, 6] or [0, nan].

Only an option when the plotType property is set to 'scatter'.

- **xScale** ↑

Sets the scale of the x-axis. Either {'linear'} | 'log'.

Only an option when plotType is set to 'scatter'.

- **xTickFrequency** ↑

Sets the frequency of the x-axis tick mark labels. Only necessary if the wanted frequency differs from the data plotted. Must be a string. One of the following

```
> 'yearly'  
> 'semiannually'  
> 'quarterly'  
> 'monthly'  
> 'weekly'  
> 'daily'
```

Caution : It is only possible to use an frequency which is lower then the frequency of the data.

Caution : When the data are of a daily frequency the default is to set the xTickFrequency to 'yearly'.

- **xTickLabelAlignment** ↑

The alignment of the x-axis tick marks labels. {'normal'} | 'middle'.

- **xTickLabelLocation** ↑

The location of the x-axis tick marks labels. Either {'bottom'} | 'top' | 'baseline'. 'baseline' will only work in combination with the xtickLocation property set to a double with the basevalue.

- **xTickLabels** ↑

Sets the x-axis tick mark labels. To translate the default tick mark label 'Var1' and 'Var2' you can use;

```
{'Var1','Label1','Var2','Label2'}
```

If given as a empty cell, default x-axis tick marks will be used, which is default.

This property can be used to create multi-lined x-axis tick mark labels. To do this give a multi-row char to the labels you want to make multi-lined.

Inherited from superclass NB_GRAPH

- **xTickLocation** ↑

The location of the x-axis tick marks. Either {'bottom'} | 'top' | double. If given as a double the tick marks will be placed at this value (where the number represent the y-axis value).

- **xTickRotation** ↑

Sets the rotation of the x-axis tick mark labels. Must be a scalar with the number of degrees the labels should be rotated. Positive angles cause counterclockwise rotation.

- **xTickStart** ↑

Sets the start date of where to start the x-tick mark labels. Must be a string or an object which of a subclass of the nb_date class.

- **yDir** ↑

Sets the direction of the y-axis of the plot. Must be a string. Either 'normal' or 'reverse'. Default is 'normal'.

Caution: If the variablesToPlotRight is not empty this setting only sets the left y-axis direction.

- **yDirRight** ↑

Sets the direction of the right y-axis of the plot. Either 'normal' or 'reverse'. Default is 'normal'. (Only when the variablesToPlotRight property is not empty.)

- **yLabel** ↑

Sets (add) the text of the y-axis label. Must be a string.

Inherited from superclass NB_GRAPH

- **yLabelFontSize** ↑

Sets the font size of the y-axis label. Must be scalar. Default is 12.

Inherited from superclass NB_GRAPH

- **yLabelFontWeight** ↑

Sets the font weight of the y-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_GRAPH

- **yLabelInterpreter** ↑

The interpreter used for the given string given by the xLabel
property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_GRAPH

- **yLabelRight** ↑

Sets the text of the y-axis label. (Only on the right side.)
Must be a string. Takes the same font option as the yLabel
property

Inherited from superclass NB_GRAPH

- **yLim** ↑

Sets the y-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Both or one of them can be
set to nan, i.e. the default limits will be used) E.g. [0 6],
[nan, 6] or [0, nan].

Caution: If the variablesToPlotRight property is not empty
this setting only sets the left y-axis limits.

Only an option for the graph(...) and graphSubPlots() methods.
(For the graphInfoStruct(...) method use the property
GraphStruct.)

- **yLimRight** ↑

Sets the right y-axis limits of the plot. Must be 1 x 2 double
vector, where the first number is the lower limit and the
second number is the upper limit. (Only when the
variablesToPlotRight property is not empty.) (Both or one of
them can be set to nan, i.e. the default limits will be used)
E.g. [0 6], [nan, 6] or [0, nan].

Only an option for the graph(...) and graphSubPlots() methods.
(For the graphInfoStruct(...) method use the property
GraphStruct.)

- **yOffset** ↑

The y-axis tick mark labels offset from the axes

- **yScale** ↑

Sets the scale of the y-axis. {'linear'} | 'log'.(Both the left and the right axes if nothing is plotted on the right axes.)

- **yScaleRight** ↑

Sets the scale of the right y-axis. {'linear'} | 'log'. (Has only somthing to say if somthing is plotted on the right axes)

- **ySpacing** ↑

Sets the spacing between y-axis tick mark labels of the y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow the frequency of the xTickFrequency property if setted, else it will follow the frequency of the data provided. (Except when dealing with daily data, because then the default xTickFrequency is 'yearly')

Caution: If the variablesToPlotRight property is not empty this property only sets the spacing between ticks of the left y-axis.

- **ySpacingRight** ↑

Sets the spacing between y-axis tick mark labels of the right y-axis. Must be scalar.

Caution: The spacing will be the number of periods between the tick marks labels. The number of periods will follow

Methods:

- [createFigure](#)
- [get](#)
- [getCode](#)
- [getData](#)
- [getGenericOptions](#)
- [getPlottedVariables](#)
- [getTSOptions](#)
- [graph](#)
- [graphInfoStruct](#)
- [graphSubPlots](#)
- [isempty](#)
- [merge](#)
- [realTimespan](#)
- [resetDataSource](#)
- [saveData](#)
- [saveFigure](#)
- [set](#)
- [struct](#)
- [timespan](#)
- [update](#)

► **createFigure** ↑

`createFigure(obj)`

Description:

Create figure to plot in.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_GRAPH

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_graph_ts

Input:

- obj : An object of class nb_graph_ts
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_graph_ts object. Be aware that all the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_graph_ts object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#), [nb_ts](#)

Written by Kenneth S. Paulsen

► **getCode** ↑

```
[code,numOfGraphs] = getCode(obj,frameTitle,frameSubTitle,...  
footer,source,pageNumber,theme)
```

Description:

Get the latex code when including a graph in a slide object produced by an object of class nb_graph_ts.

This method is called from the nb_Slide class, when making a beamer slide in latex.

Input:

- obj : An object of class nb_graph_ts
- frameTitle : A string with the frame title of the slide.
- frameSubTitle : A string with the frame subtitle of the slide.
- footer : A cellstr with the footers of the slide
- source : A cellstr with the sources of the slide
- pageNumber : Give 1 if the page number should be included.
- theme : The beamer theme used. Default is '' (use default beamer theme).

Output:

- code : The latex code as a char.
- numOfGraphs : The number of graphs produced by the input object of class nb_graph_ts. As an integer.

Examples:

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **getData** ↑

data = getData(obj, zeroLowerBound)

Description:

Get the data of the graph

Input:

- obj : An object of class nb_graph_ts
- zeroLowerBound : Zero lower bound the fan chart of the variables 'QUA_RNFOLIO' and 'QUA_RNFOLIO_LATESTMPR'.

Output:

- data : As an nb_ts object (return as nb_cs if datesToPlot is non-empty)

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

► **getGenericOptions** ↑

```
options = getGenericOptions(obj)
```

Description:

A static method of nb_graph.

Container with the intersect of options for subclasses of nb_graph.

Input:

- none

Output:

- options : A numSettings x 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getGenericOptions(obj);
```

See also:

[set](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the plotted variables of the graph

Input:

- obj : An object of class nb_graph_ts
- forLegend : true or false (default)

When forLegend is used the variables are listed as needed for the legend, but without fake legends and patches, and duplicates are not removes.

Output:

- vars : As a cellstr

Example:

```
vars = obj.getPlottedVariables();
```

Written by Kenneth S. Paulsen

► **getTSOPTIONS** ↑

```
options = getTSOPTIONS(obj)
```

Description:

Container with the properties specific to nb_graph_ts

Input:

- none

Output:

- options : A numSettings name 4 cell array with names in first column, default inputs in second, logical checks in the third, and a function handle with the set function in the fourth.

Examples:

```
options = getTSOPTIONS(obj);
```

See also:

[set](#), [getGenericOptions](#), [nb_parseInputs](#)

Written by Per Bjarne Bye

► **graph** ↑

```
graph(obj)
```

Description:

Create a graph or more graphs

Graphs all the variables given by the variablesToPlot and the variablesToPlotRight properties in one plot, and does that for all the datasets of the DB property. I.e. one new plot for each dataset (page) of the DB property of the nb_graph_ts object.

Input:

- obj : An object of class nb_graph_ts

Output:

The graph plotted on the screen.

Examples:

```
data = nb_ts([2;1;2],'''2012Q1','Var1');  
obj = nb_graph_ts(data);  
obj.graph();
```

See also:

[nb_graph_ts.set](#)

Written by Kenneth S. Paulsen

► **graphInfoStruct ↑**

```
graphInfoStruct(obj)
```

Description:

Create graphs based on the property GraphStruct. (Or the default GraphStruct given by the method defaultGraphStruct)

This method makes it easy to set up graph packages. For example graphing output from models.

See the documentation NB toolbox for more on this method.

Input:

- obj : An object of class nb_graph_ts, where the property GraphStruct contains the information on how to plot the objects data.

Output:

The wanted graphs plotted on the screen.

Examples:

```
obj.graphInfoStruct();
```

Written by Kenneth S. Paulsen

► graphSubPlots ↑

```
graphSubPlots(obj)
```

Description:

Create graphs of all the variables given in variablesToPlot property, in different subplots. (Each dataset against each other). The number of subplot is given by the property subPlotSize.

Input:

- obj : An object of class nb_graph_ts

Output:

The wanted graphs plotted on the screen.

Examples:

```
data = nb_ts([2 3 2;1 4 5;2 3 2],'', '2012Q1',...
             {'Var1','Var2','Var3'});
obj = nb_graph_ts(data);
obj.graphSubPlots();
```

Written by Kenneth S. Paulsen

► isempty ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_graph_ts object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_graph_ts

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

► merge ↑

```
out = merge(obj,varargin)
```

Description:

Default: This tries to merge the data of plotter objects that uses the graphSubPlots, or the graphInfoStruct.

Caution: All objects must share the exact same variables!

'graph' is given: In this case it will try to merge the data of all nb_graph objects, and concatenate the variableToPlot property. If the property variableToPlot is empty for all objects, it will also be returned as empty, i.e. plot all series in the data.

Caution : Merging of any of the other properties then DB and variableToPlot is not done!

Input:

- obj : An object of class nb_graph, which includes the classes nb_graph_ts, nb_graph_cs and nb_graph_data.

Optional input:

- 'graph' : Give this input to merge the graphs in the way that is described in the description of this method.
- varargin : Each input must be an object of class nb_graph.

Output:

- out : An object of class nb_graph.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_GRAPH

► **realTimespan** ↑

```
dateString = realTimespan(obj,language,freq)
```

Description:

Get the timespan of the data behind a graph which is not nan or infinite, as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_ts
- language : 'english' or 'norwegian'. 'norwegian' is default
- freq : The wanted frequency of the timespan

Output :

- dateString : A string with the timespan of the graph represented by this object.

Examples:

```
data      = nb_ts([2;1;2],'', '2012Q1','Var1');  
obj      = nb_graph_ts(data);  
dateString = realTimespan(obj);  
dateStringE = realTimespan(obj,'english');
```

Written by Kenneth S. Paulsen

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_graph_ts object.

Input :

- obj : An object of class nb_graph_ts
- dataSource : An nb_ts object with the new data.
- updateProps : If the variablesToPlot should be updated to be equal to dataSource.variables + the startGraph and endGraph properties are set to the startDate and endDate of the new data source.

Can also be 'gui'. Then all properties will be checked

against the new dataset, and if the new dataset does not comply with the properties, the properties will be updated!

Written by Kenneth S. Paulsen

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_graph_ts
- filename : A string with the saved output name
- strip : - 'on' : Strip all observation dates where all the variables has no value. Default.
 - 'off' : Does not strip all observation dates where all the variables has no value

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to (a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_graph
- extraName : If you want to add some extra name to the saveName property of the nb_graph_data object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full

```
        name of the output file.)  
  
- format      : The format of the saved output. Default is given by  
    the property fileFormat of the nb_graph. Either  
    'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.
```

Output:

The produced figure(s) saved to the wanted formats.

Examples:

```
data = nb_data([2;1;2],'', '1', 'Var1');  
obj = nb_graph_data(data);  
obj.graph();  
saveFigure(obj,'test','pdf');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_GRAPH

► **set ↑**

```
set(obj,varargin)
```

Description:

Set properties of the nb_graph class and its subclasses.

See documentation NB Toolbox or type help('nb_graph') for more on the properties of the class.

Input:

```
- obj       : An object that is a subclass of nb_graph  
- varargin : 'propertyName',PropertyValue,...
```

Output:

No actual output, but the input objects properties will have been set to their new value.

Examples:

```
data = nb_ts([2;1;2],'', '2012Q1', 'Var1');  
obj = nb_graph_ts(data);  
obj.set('startGraph','2012Q1','endGraph','2012Q3');  
obj.set('crop',1);
```

Written by Per Bjarne Bye

Inherited from superclass NB_GRAPH

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct

Input:

- obj : An object of class nb_graph_ts

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **timespan** ↑

```
dateString = timespan(obj,language,freq)
```

Description:

Get the timespan of the graph as a string. On the PPR format.

Input :

- obj : An object of class nb_graph_ts
- language : 'english' or 'norwegian'. 'norwegian' is default
- freq : The wanted frequency of the timespan

Output :

- dateString : A string with the timespan of the graph represented by this object.

Examples:

```
data      = nb_ts([2;1;2],'', '2012Q1', 'Var1');  
obj      = nb_graph_ts(data);  
dateString = timespan(obj);  
dateStringE = timespan(obj,'english');
```

Written by Kenneth S. Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the figure

See the update method of the nb_ts classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_graph_ts

Output:

- obj : An object of class nb_graph_ts, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

■ **nb_numbering**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_numbering(start,chapter,language)
```

Superclasses:

handle

Description:

A class for numbering graphs used in the nb_graph_package class.

Constructor:

```
obj = nb_numbering(start,chapter,language)
```

Input:

- start : The start number of the numbering of the figures. Must be a double.
- chapter : The chapter of the graph numbering

```
- language : The language of the graph numbering.  
          > 'norwegian' : 'Figur 1.1' or 'Figure 1.1a'.  
          > 'english'   : 'Chart 1.1' or 'Chart 1.1a'.  
  
- bigLetter : Logical that indicates if "number" should be given by  
              letters or numbers.
```

Output

```
- obj       : An object of class nb_numbering
```

Examples:

```
obj = nb_numbering(1,1,'norwegian',false)
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- bigLetter
- chapter
- counter
- language
- letter

- number

- **bigLetter** ↑

Give 1 the numbering should be done with a big letter instead of a number. I.e. 'Figur 1.A' or 'Chart 1.A', instead of 'Figure 1.1' or 'Chart 1.1'.

- **chapter** ↑

The chapter of the graph numbering. As a double.

- **counter** ↑

Indicator of letter counting. Set to 0 if the new graphs should be numbered with letters instead of normal numbering. I.e. 1a, 1b,... Otherwise 1.

- **language** ↑

The language of the graph numbering. As a string.

- **letter** ↑

The current letter of the graph numbering. As a double.

- **number** ↑

The current graph number. As a double.

Methods:

- [char](#)
 - [charData](#)
 - [charNumOnly](#)
 - [charSlide](#)
 - [figNumsAsCell](#)
 - [hold](#)
 - [plus](#)
-

► **char** ↑

```
s = char(obj,table)
```

Description:

Get the graph number represented by the nb_numbering object as a string on the format:

```
> 'norwegian' : 'Figur 1.1' or 'Figur 1.1a'.  
> 'english'   : 'Chart 1.1' or 'Chart 1.1a'.
```

Input:

- obj : An object of class nb_numbering
- table : Indicate if a table is numbered.

Output:

- s : A string on the format 'Figur 1.1', 'Figur 1.1a', 'Chart 1.1', 'Chart 1.1a'.

See also:

[charData](#)

Written by Kenneth Sæterhagen Paulsen

► **charData** ↑

```
s = charData(obj)
```

Description:

Get the graph number represented by the nb_numbering object as a string on the format: 'Data 1.1' or 'Data 1.1a'.

Input:

- obj : An object of class nb_numbering

Output:

- s : A string on the format 'Data 1.1' or 'Data 1.1a'.

Examples:

See also:

[char](#)

Written by Kenneth Sæterhagen Paulsen

► **charNumOnly** ↑

s = charOnly(obj)

Description:

Get the graph number represented by the nb_numbering object as a string on the format: '1.1' or '1.1a'.

Input:

- obj : An object of class nb_numbering

Output:

- s : A string on the format '1.1' or '1.1a'.

See also:

[char](#), [charData](#), [figNumAsCell](#)

Written by Kenneth Sæterhagen Paulsen

► **charSlide** ↑

s = charSlide(obj)

Description:

Get the slide number represented by the nb_numbering object as a string on the format: 'Slide 1.1' or 'Slide 1.1a' in english and 'Lysbilde 1.1' or 'Lysbilde 1.1a' på norsk.

Input:

- obj : An object of class nb_numbering

Output:

- s : A string on the format 'Slide 1.1' or 'Slide 1.1a' in english and 'Lysbilde 1.1' or 'Lysbilde 1.1a' på norsk.

Examples:

See also:

[char](#)

Written by Kenneth Sæterhagen Paulsen

► **figNumsAsCell ↑**

cellOut = figNumsAsCell(obj,len)

Description:

Get incrementing figure numbers as a len x 1 cell.

Input:

- obj : An object of class nb_numbering.
- len : How many graphs/tables/figures you want to number

Output:

- cellOut : A cell with the numbers.

See also:

[char](#), [charData](#), [charNumOnly](#)

Written by Per Bjarne Bye

► **hold ↑**

hold(obj,text)

Description:

Hold the normal graph numbering and start numbering with letters instead. I.e Figur 1.1a, Figur 1.1b,....

Input:

- obj : An object of class nb_numbering
- text : Either 'on' or 'off'

Output:

- obj : The input object of class nb_numbering set to on hold.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

► **plus ↑**

plus(obj)

Description:

Increment the nb_numbering object.

Input:

- obj : An object of class nb_numbering
- num : Number of graphs ahead. As a scalar. Default is 1.

Output:

- obj : The input object incremented with one number (or letter)

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

■ **nb_table_cell**

Go to: [Properties](#) | [Methods](#)

obj = nb_table_cell(data)

Superclasses:

nb_table_data_source, nb_graph_obj, handle,

Description:

A class for displaying data stored in a cell in a table.

Constructor:

obj = nb_table_cell(data)

Input:

- data : An object of class cell

Output:

- obj : An object of class nb_table_cell

See also:

[nb_table_data_source.graph](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- DB
- columnSizes
- factor
- flip
- fontUnits
- lookUpMatrix
- page
- rowSpan
- tag
- titleFontSize
- userData
- xLabelFontWeight
- a4Portrait
- columnSpan
- figureColor
- fontName
- horizontalAlignment
- noLabel
- pdfBook
- saveName
- template
- titleFontWeight
- xLabel
- xLabelInterpreter
- addAdvanced
- crop
- figureName
- fontScale
- language
- noTitle
- plotAspectRatio
- spacing
- title
- titleInterpreter
- xLabelAlignment
- xLabelPlacement
- annotation
- decimals
- figurePosition
- fontSize
- localVariables
- normalized
- position
- table
- titleAlignment
- titlePlacement
- xLabelFontSize
- xLabelPlacement

- **DB** ↑

A nbDataSource object storing the data of the table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **a4Portrait** ↑

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass NB_TABLE_DATA_SOURCE

- **addAdvanced** ↑

Set to false to prevent adding advanced components.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **annotation** ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the class for more. If you give more objects they must be collected in a cell.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSizes** ↑

Set size of each column. As a 1 numCol double.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSpan** ↑

Set columns span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two columns. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **crop** ↑

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **decimals** ↑

Round numbers to the number of decimals. Default is 4. Can either be an integer or a string with the printed format of the numbers in the table. E.g. 4, '%.4f'. See num2str for more.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **factor** ↑

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureColor** ↑

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontSize** ↑

Sets the font size of the table, default is 12. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **horizontalAlignment** ↑

Set the horizontal alignment of each cell of the table. Must be a cell with the same size as the data of the table. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noLabel** ↑

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Default is 0. This is of course only have an effect if the nb_table_data_source object have been given the xLabel properties. (These are empty by default.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the table, otherwise set it to 0. Default is 0.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **page** ↑

Sets the page of the data object to table. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_TABLE_DATA_SOURCE

- **position** ↑

Sets the position of the axes, must be an 1x4 double array. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **rowSpan** ↑

Set row span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two rows. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf'.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **spacing** ↑

Sets the spacing between the data points displayed in the table. Must be a scalar. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **table** ↑

A nb_table object storing the information of the displayed table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **template** ↑

Sets the template of the table. See nb_table.stylingPatternsTemplate for the supported templates.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'lefttaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

Methods:

- [get](#)
- [resetDataSource](#)
- [struct](#)
- [getData](#)
- [saveData](#)
- [unstruct](#)
- [getPlottedVariables](#)
- [saveFigure](#)
- [update](#)
- [graph](#)
- [set](#)
- [isempty](#)
- [setTable](#)

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_table_data_source

Input:

- obj : An object of class nb_table_data_source
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_table_data_source object. Be aware that all the the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_table_data_source object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getData ↑**

```
data = getData(obj,zeroLowerBound)
```

Description:

Get the data of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- data : As a nb_cs or nb_cell object.

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the variables of the table

Input:

- obj : An object of class nb_table_data_source

Output:

```
- vars : As a cellstr
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **graph** ↑

```
graph(obj)
```

Description:

Plot the table

Input:

- obj : An object of class nb_table_data_source

Output:

The table plotted on the screen.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_table_data_source object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_table_data_source

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_table_data_source object.

Input:

- obj : An object of class nb_table_data_source

- dataSource : An nb_dataSource object with the new data.

- updateProps : If properties should be updated or not.

Can also be 'gui'. Then all properties will be checked against the new dataset, and if the new dataset does not comply with the properties, the properties will be updated!

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

- obj : An object of class nb_table_data_source
- filename : A string with the saved output name, without extension.
- strip : - 'on' : Strip all observation dates where all the variables has no value. Default.
 - 'off' : Does not strip all observation dates where all the variables has no value

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_table_data_source
- extraName : If you want to add some extra name to the saveName property of the nb_graph_ts object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **set** ↑

`set(obj, varargin)`

Description:

Set properties of the nb_table_cell class

Input:

- varargin : Property name property value pairs.

Written by Kenneth Sæterhagen Paulsen

► **setTable** ↑

`setTable(obj, propertyName, PropertyValue)`

Description:

Set underlying table properties, which is of class nb_table.

Input:

- obj : An object of class nb_table_data_source
- propertyName : Name of the property to set.
- PropertyValue : Value of the property to set.

See also:

[nb_table](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct

Input:

- obj : An object of class nb_table_data

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

```
obj = nb_table_cell.unstruct(s)
```

Description:

Convert struct to object

Input:

- s : A struct

Output:

- obj : An object of class nb_table_cell

See also:

[nb_table_cell.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the table

See the update method of the nb_ts, nb_cs and nb_data classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_table_data_source

Output:

- obj : An object of class nb_table_data_source, where the data is updated.

Examples:

obj.update();

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

■ nb_table_cs

Go to: [Properties](#) | [Methods](#)

obj = nb_table_cs(data)

Superclasses:

nb_table_data_source, nb_graph_obj, handle,

Description:

A class for displaying time-series data in a table.

Constructor:

obj = nb_table_cs(data)

Input:

- data : An object of class nb_cs

Output:

- obj : An object of class nb_table_cs

See also:

[nb_table_data_source.graph](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [DB](#)
- [annotation](#)
- [crop](#)
- [figureColor](#)
- [flip](#)
- [fontSize](#)
- [language](#)
- [noLabel](#)
- [page](#)
- [position](#)
- [spacing](#)
- [template](#)
- [titleFontSize](#)
- [titlePlacement](#)
- [variablesOfTable](#)
- [xLabelFontSize](#)
- [xLabelPlacement](#)
- [a4Portrait](#)
- [columnSizes](#)
- [decimals](#)
- [fontName](#)
- [fontUnits](#)
- [localVariables](#)
- [noTitle](#)
- [pdfBook](#)
- [rowSpan](#)
- [table](#)
- [title](#)
- [titleFontWeight](#)
- [typesOfTable](#)
- [xLabel](#)
- [xLabelFontWeight](#)
- [addAdvanced](#)
- [columnSpan](#)
- [factor](#)
- [figurePosition](#)
- [fontScale](#)
- [horizontalAlignment](#)
- [lookUpMatrix](#)
- [normalized](#)
- [plotAspectRatio](#)
- [saveName](#)
- [tag](#)
- [titleAlignment](#)
- [titleInterpreter](#)
- [userData](#)
- [xLabelAlignment](#)
- [xLabelInterpreter](#)

- **[DB](#) ↑**

A nbDataSource object storing the data of the table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **[a4Portrait](#) ↑**

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass NB_TABLE_DATA_SOURCE

- **[addAdvanced](#) ↑**

Set to false to prevent adding advanced components.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **[annotation](#) ↑**

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the class for more. If you give more objects they must be collected in a cell.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **[columnSizes](#) ↑**

Set size of each column. As a 1 numCol double.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSpan** [↑](#)

Set columns span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two columns. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **crop** [↑](#)

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **decimals** [↑](#)

Round numbers to the number of decimals. Default is 4. Can either be an integer or a string with the printed format of the numbers in the table. E.g. 4, '%.4f'. See num2str for more.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **factor** [↑](#)

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureColor** [↑](#)

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureName** [↑](#)

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figurePosition** [↑](#)

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **flip** [↑](#)

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontSize** ↑

Sets the font size of the table, default is 12. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **horizontalAlignment** ↑

Set the horizontal alignment of each cell of the table. Must be a cell with the same size as the data of the table. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noLabel** ↑

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Default is 0. This is of course only have an effect if the nb_table_data_source object have been given the xLabel properties. (These are empty by default.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the table, otherwise set it to 0. Default is 0.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **page** ↑

Sets the page of the data object to table. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_TABLE_DATA_SOURCE

- **position** ↑

Sets the position of the axes, must be an 1x4 double array.
[leftMostPoint lowestPoint width height]. Only an option for
the graph() method. Default is [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **rowSpan** ↑

Set row span of the table. Must be a cell with the same size
as the data of the table. Give 2 to make a element span two
rows. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are
produced. Must be a string.

Default is to save each graph produced in separate files. Set
the pdfBook property to 1 if you want to save all the produced
graphs in one pdf file. (To save all figures in one file is
only possible for pdf files. I.e. the fileFormat property
must be 'pdf'.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **spacing** ↑

Sets the spacing between the data points displayed in the table.
Must be a scalar. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **table** ↑

A nb_table object storing the information of the displayed table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph
object in the subplot. Should not be used. Use userData
instead.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **template** ↑

Sets the template of the table. See
nb_table.stylingPatternsTemplate for the supported templates.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **title** ↑

Sets the title of the plot. Must be a char. Only an option
for the graph(...) method.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

> 'latex' : For mathematical expression
> 'tex' : Latex text interpreter,
> 'none' : Do nothing

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **typesOfTable** ↑

A cellstr with the types to display in the table

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **variablesOfTable** ↑

Sets the variables to be part of the table

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

Methods:

- | | | | | |
|------------|-------------------|-------------------|-----------------------|---------|
| • get | • getData | • getPlottedTypes | • getPlottedVariables | • graph |
| • isempty | • resetDataSource | • saveData | • saveFigure | • set |
| • setTable | • struct | • unstruct | • update | |

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_table_data_source

Input:

```
- obj : An object of class nb_table_data_source  
- propertyName : A string with the name of the wanted property.
```

Output:

```
- PropertyValue : The property value of the given property.
```

Examples:

Get the last axes handle of the nb_table_data_source object. Be aware that all the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_table_data_source object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getData** ↑

```
data = getData(obj, zeroLowerBound)
```

Description:

Get the data of the table

Input:

```
- obj : An object of class nb_table_data_source
```

Output:

```
- data : As a nb_cs or nb_cell object.
```

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getPlottedTypes** ↑

```
types = getPlottedTypes(obj)
```

Description:

Get the plotted types of the table

Input:

- obj : An object of class nb_table_cs

Output:

- types : As a cellstr

Example:

```
types = obj.getPlottedTypes();
```

Written by Kenneth S. Paulsen

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj,forLegend)
```

Description:

Get the variables of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- vars : As a cellstr

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **graph** ↑

```
graph(obj)
```

Description:

Plot the table

Input:

- obj : An object of class nb_table_data_source

Output:

The table plotted on the screen.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_table_data_source object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_table_data_source

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_table_data_source object.

Input:

```
- obj : An object of class nb_table_data_source  
- dataSource : An nb_dataSource object with the new data.  
- updateProps : If properties should be updated or not.  
  
Can also be 'gui'. Then all properties will be checked  
against the new dataset, and if the new dataset does not  
comply with the properties, the properties will be  
updated!
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

```
- obj : An object of class nb_table_data_source  
- filename : A string with the saved output name, without extension.  
- strip : - 'on' : Strip all observation dates where all the  
variables has no value. Default.  
- 'off' : Does not strip all observation dates where  
all the variables has no value
```

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_table_data_source
- extraName : If you want to add some extra name to the saveName property of the nb_graph_ts object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► set ↑

set(obj, varargin)

Description:

Set properties of the nb_table_cs class

Input:

- varargin : Property name property value pairs.

Written by Kenneth Sæterhagen Paulsen

► setTable ↑

setTable(obj, propertyName, PropertyValue)

Description:

Set underlying table properties, which is of class nb_table.

Input:

- obj : An object of class nb_table_data_source
- propertyName : Name of the property to set.
- PropertyValue : Value of the property to set.

See also:

[nb_table](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct

Input:

- obj : An object of class nb_table_data

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_table_cs.unstruct(s)

Description:

Convert struct to object

Input:

- s : A struct

Output:

- obj : An object of class nb_table_cs

See also:

[nb_table_cs.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the table

See the update method of the nb_ts, nb_cs and nb_data classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_table_data_source

Output:

- obj : An object of class nb_table_data_source, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

■ **nb_table_data**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_table_data(data)
```

Superclasses:

nb_table_data_source, nb_graph_obj, handle,

Description:

A class for displaying data in a table.

Constructor:

```
obj = nb_table_data(data)
```

Input:

- data : An object of class nb_data

Output:

- obj : An object of class nb_table_data

See also:

[nb_table_data_source.graph](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [DB](#)
- [annotation](#)
- [crop](#)
- [factor](#)
- [figurePosition](#)
- [fontScale](#)
- [horizontalAlignment](#)
- [lookUpMatrix](#)
- [normalized](#)
- [pdfBook](#)
- [rowSpan](#)
- [startTable](#)
- [template](#)
- [titleFontSize](#)
- [titlePlacement](#)
- [xLabel](#)
- [xLabelFontWeight](#)
- [a4Portrait](#)
- [columnSizes](#)
- [decimals](#)
- [flip](#)
- [fontSize](#)
- [language](#)
- [noLabel](#)
- [observationsOfTable](#)
- [plotAspectRatio](#)
- [saveName](#)
- [table](#)
- [title](#)
- [titleFontWeight](#)
- [userData](#)
- [xLabelAlignment](#)
- [xLabelInterpreter](#)
- [xLabelFontSize](#)
- [xLabelPlacement](#)
- [addAdvanced](#)
- [columnSpan](#)
- [endTable](#)
- [figureName](#)
- [fontName](#)
- [fontUnits](#)
- [localVariables](#)
- [noTitle](#)
- [page](#)
- [position](#)
- [spacing](#)
- [tag](#)
- [titleAlignment](#)
- [titleInterpreter](#)
- [variablesOfTable](#)
- [xLabelPlacement](#)

- [DB](#) ↑

A nbDataSource object storing the data of the table

Inherited from superclass NB_TABLE_DATA_SOURCE

- [a4Portrait](#) ↑

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass NB_TABLE_DATA_SOURCE

- [addAdvanced](#) ↑

Set to false to prevent adding advanced components.

Inherited from superclass NB_TABLE_DATA_SOURCE

- [annotation](#) ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the class for more. If you give more objects they must be collected in a cell.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSizes** [↑](#)

Set size of each column. As a 1 numCol double.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSpan** [↑](#)

Set columns span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two columns. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **crop** [↑](#)

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **decimals** [↑](#)

Round numbers to the number of decimals. Default is 4. Can either be an integer or a string with the printed format of the numbers in the table. E.g. 4, '%.4f'. See num2str for more.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **endTable** [↑](#)

Sets the end obs of the table. As an integer. If the given obs is after the end obs of the data, the data will be appended with nan values. I.e. the table will be blank for these dates.

- **factor** [↑](#)

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureColor** [↑](#)

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureName** [↑](#)

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figurePosition** [↑](#)

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **flip** [↑](#)

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontName** [↑](#)

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontScale** [↑](#)

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontSize** [↑](#)

Sets the font size of the table, default is 12. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontUnits** [↑](#)

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **horizontalAlignment** [↑](#)

Set the horizontal alignment of each cell of the table. Must be a cell with the same size as the data of the table. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noLabel** ↑

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Default is 0. This is of course only have an effect if the nb_table_data_source object have been given the xLabel properties. (These are empty by default.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the table, otherwise set it to 0. Default is 0.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **observationsOfTable** ↑

A vector with the observations to display in the table, this property will overrun the startTable and endTable properties.

- **page** ↑

Sets the page of the data object to table. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_TABLE_DATA_SOURCE

- **position** ↑

Sets the position of the axes, must be an 1x4 double array. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **rowSpan** ↑

Set row span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two rows. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf').

Inherited from superclass NB_TABLE_DATA_SOURCE

- **spacing** ↑

Sets the spacing between the data points displayed in the table. Must be a scalar. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **startTable** ↑

Sets the start obs of the graph. As an integer. If this obs is before the start obs of the data, the data will be expanded with nan (blank values) for these periods.

- **table** ↑

A nb_table object storing the information of the displayed table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **template** ↑

Sets the template of the table. See nb_table.stylingPatternsTemplate for the supported templates.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'lefttaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **variablesOfTable** ↑

Sets the variables to be part of the table

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

Methods:

- get
- getData
- getPlottedVariables
- graph
- isempty
- resetDataSource
- saveData
- saveFigure
- set
- setTable
- struct
- unstruct
- update

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_table_data

Input:

- obj : An object of class nb_table_data
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_table_data object. Be aware that all the the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_table_data object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth S. Paulsen

► **getData** ↑

```
data = getData(obj, zeroLowerBound)
```

Description:

Get the data of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- data : As a nb_cs or nb_cell object.

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the variables of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- vars : As a cellstr

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **graph** ↑

```
graph(obj)
```

Description:

Plot the table

Input:

- obj : An object of class nb_table_data_source

Output:

The table plotted on the screen.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_table_data_source object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_table_data_source

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_table_data_source object.

Input:

```
- obj : An object of class nb_table_data_source  
- dataSource : An nb_dataSource object with the new data.  
- updateProps : If properties should be updated or not.  
  
Can also be 'gui'. Then all properties will be checked  
against the new dataset, and if the new dataset does not  
comply with the properties, the properties will be  
updated!
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

```
- obj : An object of class nb_table_data_source  
- filename : A string with the saved output name, without extension.  
- strip : - 'on' : Strip all observation dates where all the  
variables has no value. Default.  
- 'off' : Does not strip all observation dates where  
all the variables has no value
```

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveFigure** ↑

```
saveFigure(obj,extraName,format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_table_data_source
- extraName : If you want to add some extra name to the saveName property of the nb_graph_ts object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► set ↑

set(obj, varargin)

Description:

Set properties of the nb_table_data class

Input:

- varargin : Property name property value pairs.

Written by Kenneth Sæterhagen Paulsen

► setTable ↑

setTable(obj, propertyName, PropertyValue)

Description:

Set underlying table properties, which is of class nb_table.

Input:

- obj : An object of class nb_table_data_source
- propertyName : Name of the property to set.
- PropertyValue : Value of the property to set.

See also:

[nb_table](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct

Input:

- obj : An object of class nb_table_data

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_table_data.unstruct(s)

Description:

Convert struct to object

Input:

- s : A struct

Output:

- obj : An object of class nb_table_data

See also:

[nb_table_data.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the table

See the update method of the nb_ts, nb_cs and nb_data classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_table_data_source

Output:

- obj : An object of class nb_table_data_source, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

■ **nb_table_ts**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_table_ts(data)
```

Superclasses:

nb_table_data_source, nb_graph_obj, handle,

Description:

A class for displaying time-series data in a table.

Constructor:

```
obj = nb_table_ts(data)
```

Input:

- data : An object of class nb_ts

Output:

- obj : An object of class nb_table_ts

See also:

[nb_table_data_source.graph](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [DB](#)
- [annotation](#)
- [crop](#)
- [endTable](#)
- [figureName](#)
- [fontName](#)
- [fontUnits](#)
- [localVariables](#)
- [noTitle](#)
- [pdfBook](#)
- [rowSpan](#)
- [startTable](#)
- [template](#)
- [titleFontSize](#)
- [titlePlacement](#)
- [xLabel](#)
- [xLabelFontWeight](#)
- [a4Portrait](#)
- [columnSizes](#)
- [datesOfTable](#)
- [factor](#)
- [figurePosition](#)
- [fontScale](#)
- [horizontalAlignment](#)
- [lookUpMatrix](#)
- [normalized](#)
- [plotAspectRatio](#)
- [saveName](#)
- [table](#)
- [title](#)
- [titleFontWeight](#)
- [userData](#)
- [xLabelAlignment](#)
- [xLabelInterpreter](#)
- [xLabelPlacement](#)
- [addAdvanced](#)
- [columnSpan](#)
- [decimals](#)
- [figureColor](#)
- [flip](#)
- [fontSize](#)
- [language](#)
- [noLabel](#)
- [page](#)
- [position](#)
- [spacing](#)
- [tag](#)
- [titleAlignment](#)
- [titleInterpreter](#)
- [variablesOfTable](#)
- [xLabelFontSize](#)
- [xLabelPlacement](#)

- [DB](#) ↑

A nbDataSource object storing the data of the table

Inherited from superclass NB_TABLE_DATA_SOURCE

- [a4Portrait](#) ↑

Save PDF to A4 portrait format. 0 or 1

Inherited from superclass NB_TABLE_DATA_SOURCE

- [addAdvanced](#) ↑

Set to false to prevent adding advanced components.

Inherited from superclass NB_TABLE_DATA_SOURCE

- [annotation](#) ↑

Sets the plotted annotations of the graph.

Must be one or more nb_annotation objects. See the documentation of the class for more. If you give more objects they must be collected in a cell.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSizes** [↑](#)

Set size of each column. As a 1 numCol double.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **columnSpan** [↑](#)

Set columns span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two columns. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **crop** [↑](#)

Sets the how the figure is saved to .pdf and other figure formats. If 1 is given the output file will be cropped. Default is not to crop (i.e. set to 0).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **datesOfTable** [↑](#)

The dates of the table, as a cellstr. This option will overrun the startTable and endTable properties.

- **decimals** [↑](#)

Round numbers to the number of decimals. Default is 4. Can either be an integer or a string with the printed format of the numbers in the table. E.g. 4, '%.4f'. See num2str for more.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **endTable** [↑](#)

Sets the end date of the table. Either as string or an object which is of a subclass of the nb_date class. If the given date is after the end date of the data, the data will be appended with nan values. I.e. the table will be blank for these dates.

- **factor** [↑](#)

Set this property to multiply the data with a given factor. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureColor** [↑](#)

Color of the figure as a 1x3 double. Default is [1 1 1].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figureName** ↑

Sets the name of figure (only in MATLAB), default is no name.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **figurePosition** ↑

Position of the figure as a 1x4 double. Default is []. I.e. use the MATLAB default. In characters.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **flip** ↑

For some reason the graph is sometimes flipped and sometimes not. If the graph is flipped use this property to flip it back.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontName** ↑

Sets the name of font used. Default is 'arial'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontScale** ↑

Sets the parameter that scales all the font sizes of the graphs. Must be scalar. Default is 1 (do not scale).

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontSize** ↑

Sets the font size of the table, default is 12. Must be a scalar.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **fontUnits** ↑

Sets the font units of all the fonts of the graph. Either {'points'} | 'normalized' | 'inches' | 'centimeters'. Default is 'points'.

Caution : If you set the fontUnits property in a method call to the set method, all the font properties set before the fontUnits property will be set in the old fontUnits, while all the font properties set after will be in the new fontUnits.

Caution : 'normalized' font units must be between 0 and 1. The normalized units are not the same as the MATLAB normalized units. This normalized units are not only normalized to the axes size, but also across resolution of the screen. (But only vertically)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **horizontalAlignment** ↑

Set the horizontal alignment of each cell of the table. Must be a cell with the same size as the data of the table. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **language** ↑

Sets the language styles of the graphs. Must be a string with either 'norsk' or 'english'. 'english' is default. I.e. axis settings differs between norwegian and english graphs.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **localVariables** ↑

A nb_struct with the local variables of the nb_graph object. I.e. a field with name 'test' can be reach with the string input %#test.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **lookUpMatrix** ↑

Sets how the given mnemonics (variable and type names) should map to different languages. Must be cell array on the form:

```
{'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Or a .m file name, as a string. The .m file must include (and only include) what follows:

```
obj.lookUpMatrix = {
'mnemonics1','englishDescription1','norwegianDescription1';
'mnemonics2','englishDescription2','norwegianDescription2'};
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noLabel** ↑

Set it to 1 if no label(s) is/are wanted on the graphs, otherwise set it to 0. Default is 0. This is of course only have an effect if the nb_table_data_source object have been given the xLabel properties. (These are empty by default.)

Inherited from superclass NB_TABLE_DATA_SOURCE

- **noTitle** ↑

Set it to 1 if you don't want title(s) of the table, otherwise set it to 0. Default is 0.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **normalized** ↑

If the font should be normalized to the figure or axes. Either 'figure' or 'axes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **page** ↑

Sets the page of the data object to table. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **pdfBook** ↑

Set it to 1 if you want store all the created graphs produced in one pdf file. Must be use in combination with the saveName property.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **plotAspectRatio** ↑

Sets the plot aspect ratio. Either [] or '[4,3]'. Default is [].

When set to '[4,3]' the aspect ratio of the figure is 4 to 3, or when '[16,9]' the aspect ratio of the figure is 16 to 9

Inherited from superclass NB_TABLE_DATA_SOURCE

- **position** ↑

Sets the position of the axes, must be an 1x4 double array. [leftMostPoint lowestPoint width height]. Only an option for the graph() method. Default is [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TABLE_DATA_SOURCE

- **rowSpan** ↑

Set row span of the table. Must be a cell with the same size as the data of the table. Give 2 to make a element span two rows. Empty cell elements are given default values.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **saveName** ↑

Sets the saved file name. If not given, no file(s) is/are produced. Must be a string.

Default is to save each graph produced in separate files. Set the pdfBook property to 1 if you want to save all the produced graphs in one pdf file. (To save all figures in one file is only possible for pdf files. I.e. the fileFormat property must be 'pdf').

Inherited from superclass NB_TABLE_DATA_SOURCE

- **spacing** ↑

Sets the spacing between the data points displayed in the table. Must be a scalar. Default is 1.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **startTable** ↑

Sets the start table of the graph. Must be a string or an object which of a subclass of the nb_date class. If this date is before the start date of the data, the data will be expanded with nan (blank values) for these periods.

- **table** ↑

A nb_table object storing the information of the displayed table

Inherited from superclass NB_TABLE_DATA_SOURCE

- **tag** ↑

Used by the nb_graph_subplot class to locate the nb_graph object in the subplot. Should not be used. Use userData instead.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **template** ↑

Sets the template of the table. See nb_table.stylingPatternsTemplate for the supported templates.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **title** ↑

Sets the title of the plot. Must be a char. Only an option for the graph(...) method.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleAlignment** ↑

Sets the figure title text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontSize** ↑

Sets the font size of the titles which is placed above the (sub)plot(s). Must be scalar. Default is 14.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleFontWeight** ↑

Sets the font weight of the titles which is placed above the (sub)plot(s). Must be a string. Default is 'bold'.

Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titleInterpreter** ↑

Sets the interpreter used for the given string given by the title property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **titlePlacement** ↑

Sets the placement of the title. {'center'} | 'left' | 'right' | 'lefttaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **userData** ↑

User data, can be set to anything you want.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **variablesOfTable** ↑

Sets the variables to be part of the table

- **xLabel** ↑

Sets (add) the text of the x-axis label. Must be a string.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelAlignment** ↑

Sets the x-axis label text alignment, default is 'center'. You also have 'left' and 'right'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontSize** ↑

Sets the font size of the x-axis label. Must be scalar.
Default is 12.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelFontWeight** ↑

Sets the font weight of the x-axis label. Must be string.
Either 'bold', 'normal', 'demi' or 'light'.

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelInterpreter** ↑

The interpreter used for the given string given by the xlabel property.

```
> 'latex' : For mathematical expression  
> 'tex'   : Latex text interpreter,  
> 'none'  : Do nothing
```

Inherited from superclass NB_TABLE_DATA_SOURCE

- **xLabelPlacement** ↑

Sets the placement of the x-axis label. {'center'} | 'left' | 'right' | 'leftaxes'.

Inherited from superclass NB_TABLE_DATA_SOURCE

Methods:

- | | | | | |
|----------------|-------------------|-----------------------|--------------|-----------|
| • get | • getData | • getPlottedVariables | • graph | • isempty |
| • realTimespan | • resetDataSource | • saveData | • saveFigure | • set |
| • setTable | • struct | • timespan | • unstruct | • update |

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get some otherwise non-accessible properties of the nb_table_ts

Input:

- obj : An object of class nb_table_ts
- propertyName : A string with the name of the wanted property.

Output:

- propertyValue : The property value of the given property.

Examples:

Get the last axes handle of the nb_table_ts object. Be aware that all the the axes handles are stored in the figurehandle property of the object.

```
axeshandle = obj.get('axeshandle');
```

Get the handle of all the current figures of the nb_table_ts object.

```
figurehandle = obj.get('figurehandle');
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth S. Paulsen

► **getData** ↑

```
data = getData(obj, zeroLowerBound)
```

Description:

Get the data of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- data : As a nb_cs or nb_cell object.

Example:

```
data = obj.getData();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **getPlottedVariables** ↑

```
vars = getPlottedVariables(obj, forLegend)
```

Description:

Get the variables of the table

Input:

- obj : An object of class nb_table_data_source

Output:

- vars : As a cellstr

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **graph** ↑

```
graph(obj)
```

Description:

Plot the table

Input:

- obj : An object of class nb_table_data_source

Output:

The table plotted on the screen.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Test if an nb_table_data_source object is empty. I.e. if no data is stored in the object. Return 1 if true, otherwise 0.

Input:

- obj : An object of class nb_table_data_source

Output:

- ret : True (1) if the series isempty, false (0) if not

Examples:

```
ret = isempty(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **realTimespan** ↑

```
dateString = realTimespan(obj,language,freq)
```

Description:

Get the timespan of the data behind a table which is not nan or infinite, as a string. On the PPR format.

Input :

- obj : An object of class nb_table_ts

```
- language : 'english' or 'norwegian'. 'norwegian' is default  
- freq : The wanted frequency of the timespan  
  
Output :  
  
- dateString : A string with the timespan of the graph  
represented by this object.
```

Written by Kenneth S. Paulsen

► **resetDataSource** ↑

```
[message,err] = resetDataSource(obj,dataSource,updateProps)
```

Description:

Reset the data source of the nb_table_data_source object.

Input:

```
- obj : An object of class nb_table_data_source  
- dataSource : An nb_dataSource object with the new data.  
- updateProps : If properties should be updated or not.
```

Can also be 'gui'. Then all properties will be checked
against the new dataset, and if the new dataset does not
comply with the properties, the properties will be
updated!

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveData** ↑

```
obj = saveData(obj,filename,strip)
```

Description:

Saves the data of the figure

Input:

```
- obj : An object of class nb_table_data_source  
- filename : A string with the saved output name, without extension.
```

```
- strip      : - 'on'   : Strip all observation dates where all the
                  variables has no value. Default.

                  - 'off'  : Does not strip all observation dates where
                  all the variables has no value
```

Example:

```
obj.saveData('test');
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **saveFigure** ↑

```
saveFigure(obj, extraName, format)
```

Description:

Save the figure(s) produced by the nb_graph object to
(a) file(s) on the wanted format and name(s).

Input:

- obj : An object of class nb_table_data_source
- extraName : If you want to add some extra name to the saveName property of the nb_graph_ts object before saving the file. (If the saveName property of the nb_graph_ts object is empty this will be the full name of the output file.)
- format : The format of the saved output. Default is given by the property fileFormat of the nb_graph. Either 'eps', 'jpg', 'pdf', 'png', 'svg' or 'emf'.

Output:

The produced figure(s) saved to the wanted formats.

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **set** ↑

```
set(obj, varargin)
```

Description:

Set properties of the nb_table_ts class

Input:

- varargin : Property name property value pairs.

Written by Kenneth Sæterhagen Paulsen

► **setTable** ↑

`setTable(obj,propertyName,value)`

Description:

Set underlying table properties, which is of class nb_table.

Input:

- obj : An object of class nb_table_data_source
- propertyName : Name of the property to set.
- propertyValue : Value of the property to set.

See also:

[nb_table](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

► **struct** ↑

`s = struct(obj)`

Description:

Convert object to struct

Input:

- obj : An object of class nb_table_ts

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **timespan** ↑

```
dateString = timespan(obj,language,freq)
```

Description:

Get the timespan of the table as a string. On the PPR format.

Input :

- obj : An object of class nb_table_ts
- language : 'english' or 'norwegian'. 'norwegian' is default
- freq : The wanted frequency of the timespan

Output :

- dateString : A string with the timespan of the graph represented by this object.

Written by Kenneth S. Paulsen

► **unstruct** ↑

```
obj = nb_table_ts.unstruct(s)
```

Description:

Convert struct to object

Input:

- s : A struct

Output:

- obj : An object of class nb_table_ts

See also:

[nb_table_ts.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj)
```

Description:

Update the data source of the table

See the update method of the nb_ts, nb_cs and nb_data classes for more on how to make the data of the graph updateable.

Input:

- obj : An object of class nb_table_data_source

Output:

- obj : An object of class nb_table_data_source, where the data is updated.

Examples:

```
obj.update();
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_TABLE_DATA_SOURCE

◆ **nb_combineGraphStructDensityGraphs**

```
plotter = nb_combineGraphStructDensityGraphs(varargin)
```

Description:

Inputs are nb_graph_ts objects where each object uses the fanDatasets property in combination with GraphStruct property.

Input:

- varargin : Every second input starting with the first must be the a string (Used by the legend), and every second element starting with the second element must be an object of class nb_graph_ts.

Output:

- plotter : An object you can use the graphInfoStruct method or the nb_graphInfoStructGUI class on.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_getDataNumbering**

```
[numStr,counter] = nb_getFigureNumbering(graphObj,numberObj,counter)
```

Description:

Help function for numbering graphs.

Caution : numberObj is a nb_numbering object handle, and therefore need
not be returned!

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getFigureNumbering**

```
[numStr,counter] = nb_getFigureNumbering(graphObj,numberObj,counter)
```

Description:

Help function for numbering graphs.

Caution : numberObj is a nb_numbering object handle, and therefore need
not be returned!

Written by Kenneth Sæterhagen Paulsen

◆ **nb_graph_init**

```
obj = nb_graph_init(data)
```

Description:

A function to create a corresponding graph object to the provided data
object

Input:

- data : Either an object of class nb_ts, nb_cs or nb_data.

Output:

- obj : Either an object of class nb_graph_ts, nb_graph_cs and
nb_graph_data

Written by Kenneth Sæterhagen Paulsen

◆ **nb_localFunction**

```
outString = nb_localFunction(obj,inString)
```

Description:

Finds the local function used in a char or a cellstr, i.e. a word starting with %#obj., and substitutes it with the matching method of the provided object.

Input:

- obj : A NB toolbox object.
- inString : A char or a cellstr.

Output:

- outString : A char or a cellstr. (Will match the input)

Written by Kenneth Sæterhagen Paulsen

◆ nb_localVariables

```
outString = nb_localVariables(localVariables,inString)
```

Description:

Finds the local variables used in a char or a cellstr, i.e. a word starting with %#, and substitutes it with the matching field of the localVariables input

Input:

- localVariables : A MATLAB struct or an nb_struct object.
- inString : A char or a cellstr.

Output:

- outString : A char or a cellstr. (Will match the input)

Written by Kenneth Sæterhagen Paulsen and Henrik Halvorsen Hortemo

◆ nb_plots

```
nb_plots(structure,varargin)
plotter = nb_plots(structure,varargin)
```

Description:

Plots a structure of nb_ts objects.

This function plots each variable of each nb_ts objects in separate plots. If the nb_ts objects have more datasets (pages) it will plot these against each other.

This function can be utilized to plot impulse response functions returned by the nb_irf function.

Input:

- structure : A structure of nb_ts objects.

Optional input (...,'propertyName',PropertyValue,...):

- varargin : - See the optional input to the set method of the nb_graph_ts class.

- 'gui' : Give 1 if the one figure with menu to select the graphs is wanted, else 0. Default is 0.

Caution: The saveName property will be disabled.

- 'translate' :

A cellstr array with how to translate the given variables. I.e. {{'modelName1','translation1'},...}

- 'parent' : Parent given to the nb_graphInfoStructGUI class when 'gui' is set to 1. Default is [].

Output:

- plotter : A nb_graph_ts object. Use the graphInfoStruct method to produce graphs.

- If nargout is 0:

Graphs of each variable of each nb_ts objects in seperate plots on the screen. If the nb_ts objects have more datasets (pages) it will plot these against eachother.

Examples:

If you provide the 'saveName' and the 'pdfBook' options all the plotted variables will be saved in one pdf file.

```
nb_plots(structure,'saveName','test','pdfBook',1);
```

Written by Kenneth Sætherhagen Paulsen

◆ nb_saveas

```
nb_saveas(gcf,saveName)
nb_saveas(fig,saveName,format)
nb_saveas(fig,saveName,format,'-noflip')
nb_saveas(fig,saveName,'pdf','nocrop') % Only for pdf and jpg
nb_saveas(fig,saveName,'pdf','-append') % Only for pdf
```

Description:

Saves the provided figureHandle to the provided file format.

Uses the export_fig package and Ghostscript.

Input:

- fig : Handle to the current figure
- saveName : The save file name. With or without extension.
- format : The saved file format:
 - 'pdf' : PDF file using the export_fig
 - 'emf' : EMF using print
 - 'eps' : EPS using print
 - 'png' : PNG file using the export_fig
 - 'dpng' : PNG using print
 - 'svg' :
 - 'jpg' : JPG file using the export_fig
 - 'djpeg' : JPG using print

Optional input:

- '-noflip' : Not flip the saved output.
- '-nocrop' : Don't crop pdf figures.
- '-append' : Append to existing pdf.
- '-a4portrait' : A4 portrait format. Will overrule the other optional inputs.

Output:

- The figure saved to the wanted file format.

Examples:

See also:

[saveas](#)

Written by Kenneth Sæterhagen Paulsen

25.4 Basic graphics classes and functions

- nb_alpha
- nb_arrow
- nb_bar
- nb_breakText
- nb_candle
- nb_colormap
- nb_defaultColors
- nb_donut
- nb_drawCircle
- nb_drawPatch
- nb_figure
- nb_footer
- nb_getFanColors
- nb_gradedFanChart
- nb_hbar
- nb_horizontalLine
- nb_image
- nb_isAxes
- nb_isContextMenu
- nb_isPanel
- nb_isUipanel
- nb_ismarker
- nb_legend
- nb_line
- nb_pie
- nb_plotBarAtEnd
- nb_plotLabels
- nb_area
- nb_axes
- nb_barAnnotation
- nb_breakTextAfterEdit
- nb_colorbar
- nb_curlyBrace
- nb_displayValue
- nb_dpos2dpos
- nb_drawLine
- nb_fanChart
- nb_figureTitle
- nb_getCurrentPointInAxesUnits
- nb_getFontSize
- nb_graphPanel
- nb_highlight
- nb_htmlColors
- nb_interpretKeyPress
- nb_isColorProp
- nb_isFigure
- nb_isRGB
- nb_islineStyle
- nb_istype
- nb_legendDetails
- nb_patch
- nb_plot
- nb_plotComb
- nb_pos2pos

- nb_radar
 - nb_scaleLineWidth
 - nb_subplot
 - nb_subplot_position
 - nb_textArrow
 - nb_title
 - nb_verticalLine
 - nb_xlabel
 - nb_regressLine
 - nb_setDefinedColorAnnotation
 - nb_subplotSpecial
 - nb_table
 - nb_textBox
 - nb_unitsRatio
 - nb_wrapFigureText
 - nb_ylabel
-

■ nb_area

Go to: [Properties](#) | [Methods](#)

```
obj = nb_area(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making area plots.

The handle have set and get methods as all MATLAB graph handles.

Caution: All the children of this object is of class nb_patch (or line)

Constructor:

```
obj = nb_area(xData,'propertyName',PropertyValue,...)
obj = nb_area(xData,yData,'propertyName',PropertyValue,...)
```

Input:

- xData : The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)
- yData : The yData of the plot. The columns are counted as seperate variables
- varargin : ..., 'propertyName', PropertyValue, ...

Output:

- obj : An object of class nb_area

Examples:

```
obj = nb_area([1,2]); % Provide only y-axis data
obj = nb_area([1,2],[1,2]);
obj = nb_area([1,2],'propertyName',PropertyValue,...)
obj = nb_area([1,2],[1,2],'propertyName',PropertyValue,...)
```

See also:

`area`

Written by Kenneth Sætherhagen Paulsen

Properties:

- `abrupt`
- `accumulate`
- `baseValue`
- `baseline`
- `cData`
- `children`
- `deleteOption`
- `faceAlpha`
- `legendInfo`
- `lineStyle`
- `lineWidth`
- `parent`
- `side`
- `sumTo`
- `visible`
- `xData`
- `yData`

- **abrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan.

- **accumulate** ↑

Set if the areas should be accumulated or not. Default is true.

- **baseValue** ↑

The base value of the area and bar plot. Must be a scalar. Default is 0.

- **baseline** ↑

An nb_horizontalLine handle (object). Use the set and get methods of this handle to change the baseline properties

- **cData** ↑

The color data of the line plotted. Must be of size; `size(yData, 2) x 3` with the RGB colors or a cellstr with size `1 x size(yData, 2)` with the color names.

- **children** ↑

All the handles of the area plot, as patch objects

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **faceAlpha** ↑

Transparency of the patch face. Only the scalar option is supported. A single non-NaN value between 0 and 1 that controls the transparency of all the faces of the object. 1 (the default) means fully opaque and 0 means completely transparent (invisible)

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

The line style of the edge of the areas. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | ':' | '-' | 'none' .

- **lineWidth** ↑

The line width of the edge of the area. As a scalar. Default is 1

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes (Which is the default).

- **sumTo** ↑

If given the sum of the bars for each period will sum up to this number (only when style is set to 'stacked'). E.g. if set 100, each bar for each variable will be given as percentage share of the total sum.

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'

- **xData** ↑

The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)

- **yData** ↑

The yData of the plot. The columns are counted as separate variables

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get ↑**

propertyValue = get(obj,propertyName)

Description:

Get a property of an object of class nb_area

Input:

- obj : An object of class nb_area
- propertyName : A string with the propertyName

Output:

- `propertyValue` : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the `nb_plotHandle` class.

Get the default colors given the number of plotted variables.

Input:

- `dataSize` : Number of plotted variables.

Output:

- `defaultColors` : The default colors, as a $M \times 3$ double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_PLOTHANDLE`

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the `nb_plotHandle` class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blå','b'
```

```
> 'red','rÃ,d','r'  
> 'green','grÃ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÃ¥','lb'  
> 'black','svart','k'  
> 'grey','grÃ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÃ¥','hb'  
> 'deep blue','mÃ,rk blÃ¥','db'  
> 'water blue','vannblÃ¥','vb'  
> 'cool grey','kald grÃ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_area. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_area
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_area with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth SÃ¶terhagen Paulsen

■ nb_arrow

Go to: [Properties](#) | [Methods](#)

`nb_lineAnnotation, nb_movableAnnotation, nb_annotation, handle`

Description:

A class for adding arrows to a plot.

Constructor:

```
obj = nb_arrow(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_arrow

Examples:

See also:

[nb_annotation](#), [handle](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth SÃ¶terhagen Paulsen

Properties:

- | | | | | |
|---------------|--------------|--------------|---------------|----------------|
| • ID | • children | • color | • copyOption | • deleteOption |
| • head1Length | • head1Style | • head1Width | • head2Length | • head2Style |
| • head2Width | • lineStyle | • lineWidth | • listeners | • parent |
| • pointer | • selected | • side | • units | • xData |
| • yData | | | | |

- [ID](#) ↑

`nb_arrow/ID` is a property.

- **children** ↑

The children of the handle.

Inherited from superclass `NB_ANNOTATION`

- **color** ↑

The color data of the line plotted. Must be of size;
1 x 3. With the RGB colors. Or a string with the
name of the color. See the method `interpretColor`
of the `nb_plotHandle` class for more on the supported
color names.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu
of annotation object. Default is false.

Inherited from superclass `NB_ANNOTATION`

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be
deleted and all that is plotted by this object are
removed from the figure it is plotted on. If on the other
hand 'only' is given, it will just delete the object

Inherited from superclass `NB_ANNOTATION`

- **head1Length** ↑

Length of first arrowhead. Size in points. As a scalar.
Specify in points.1 point = 1/72 inch. The default
value is 8.

The first arrowhead is located at the start defined by
the point `xData(1)`, `yData(1)`.

- **head1Style** ↑

Style of first arrowhead. As a string. Specify this
property as one of the strings from the following table.

```
> 'none'  
> 'plain'  
> 'ellipse'  
> 'vback1'  
> 'vback2' (default)  
> 'vback3'  
> 'cback1'  
> 'cback2'  
> 'cback3'  
> 'star4'  
> 'rectangle'  
> 'diamond'
```

```
> 'rose'  
> 'hypocycloid'  
> 'astroid'  
> 'deltoid'
```

- **head1Width** ↑

Width of first arrowhead. Specify in points.
1 point = 1/72 inch. As a scalar. The default value is
8.

The first arrowhead is located at the start defined by
the point `xData(1)`, `yData(1)`.

- **head2Length** ↑

Length of second arrowhead. Size in points. As a scalar.
Specify in points. 1 point = 1/72 inch. The default value
is 10.

The second arrowhead is located at the end defined by
the point `xData(2)`, `yData(2)`.

- **head2Style** ↑

Style of second arrowhead. As a string. Specify this
property as one of the strings from the following table.

```
> 'none' (default)  
> 'plain'  
> 'ellipse'  
> 'vback1'  
> 'vback2'  
> 'vback3'  
> 'cback1'  
> 'cback2'  
> 'cback3'  
> 'star4'  
> 'rectangle'  
> 'diamond'  
> 'rose'  
> 'hypocycloid'  
> 'astroid'  
> 'deltoid'
```

- **head2Width** ↑

Width of second arrowhead. Specify in points.
1 point = 1/72 inch. As a scalar. The default value is
10.

The second arrowhead is located at the end defined by
the point `xData(2)`, `yData(2)`.

- **lineStyle** ↑

The line style of the plotted data. Must be a string.
{'-' } | '--' | ':' | '-' | 'none'. '---' is not supported.

- **lineWidth** ↑

The line width of the plotted arrow. As an integer.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

The axes to plot against. 'left' or 'right'. 'left' is default.

- **units** ↑

The coordinates system to plot in. Either {'data'} :
xy-coordinates | 'normalized' : figure coordinates.

- **xData** ↑

The x-axis data. As a 1x2 double. E.g: [x_begin, x_end].

- **yData** ↑

The y-axis data. As a 1x2 double. E.g: [y_begin, y_end].

Methods:

- [set](#)

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_arrow. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_arrow
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_arrow with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_axes

Go to: [Properties](#) | [Methods](#)

```
obj = nb_axes(varargin)
```

Superclasses:

handle

Description:

This is a class for making plot axes.

This class does not supports all the normal axes properties through the set and get methods of this class, but the most important ones.

But it is possible to use the set and get methods of the MATLAB axes class on the properties axesHandle and axesHandleRight of this class after constructing a nb_axes object.

- This class supports y-axes on both sides
- Some extra added functionalites:

```
> Shaded background  
> Set the space between each tick marks of both the  
y-axis and x-axis
```

Caution : This way of making axes is not stable when using the normal MATLAB plotting commands, so I recommend to use the "nb" plotting classes:

```
> nb_area  
> nb_bar  
> nb_hbar  
> nb_highlight  
> nb_horizontalLine  
> nb_line  
> nb_patch  
> nb_pie  
> nb_plot  
> nb_radar  
> nb_scatter  
> nb_plotComb  
> nb_verticalLine
```

I also strongly recommend to use the nb_axes object as the parent of the listet classes above. The default is to use a nb_axes object as the parent for these classes.

The axes will automatically adjust given the children of the nb_axes object. But it is also possible to freeze the axes options if you set the update property to 'off'.

Titles of axes:

You must use the handle you get when intializing the nb_axes object as the parent property of the nb_title class. I.e:

```
> ax = nb_axes();  
    nb_title('A title','parent',ax)  
  
> nb_title('A title') WILL NOT WORK!
```

Lables on the axes:

You must use the handle you get when intializing the nb_axes object as the parent property of the nb_yLabel class. I.e:

```
> ax = nb_axes();  
    nb_yLabel('A y-axis label','parent',ax)  
  
> nb_yLabel('A y-axis label') WILL NOT WORK!
```

It is also possible to decide which axis to place the label on, i.e. 'right' or 'left', by the property side. For more see the documentation of the nb_ylabel class.

The xlabel can be set in the same way as the ylabel, but you do not have the side option.

Constructor:

```

nb_axes
nb_axes('propertyName',PropertyValue,...)
ax = nb_axes('propertyName',PropertyValue,...)

Input:

- varargin : ..., 'propertyName', PropertyValue, ...

Support most the inputs to the MATLAB axes
class for the left axes, but some extra
options is available:

Right axis options:
> 'yDirRight'      : Direction of the axes on
                      the right
> 'yLabelRight'    : The y-axis label of the
                      plot
> 'yLimRight'      : y-axis limit of the right
                      axes

Shaded background options:
> 'shading' : {'none'} or 'grey'

```

Output:

- obj : A nb_axes handle (object). Which can be given as the parent for the nb_* plotting classes

A plotted axes in the current figure or newly made figure

Examples:

```

ax = nb_axes('position',[0.13 0.11 0.775 0.815],...
             'xLim',[0,1],'yLim',[0.1]);

```

same as

```

ax = nb_axes

```

See also:

[axes](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- `UIContextMenu`
- `annotations`
- `axesHandleRight`
- `axisVisible`
- `color`
- `dashType`
- `fast`
- `fontName`
- `fontSizeX`
- `fontWeight`
- `gridLineStyle`
- `horizontalLine`
- `language`
- `limMode`
- `normalized`
- `parent`
- `plotAxesHandleRight`
- `position`
- `scaleFactor`
- `shading`
- `tickDir`
- `title`
- `update`
- `visible`
- `xLim`
- `xOffset`
- `xTick`
- `alignAxes`
- `axesHandle`
- `axesLabelHandle`
- `children`
- `colorMap`
- `deleteOption`
- `findAxisLimitMethod`
- `fontSize`
- `fontUnits`
- `grid`
- `highLighted`
- `imageChild`
- `legend`
- `lineWidth`
- `orientation`
- `plotAxesHandle`
- `plotBoxAspectRatio`
- `precision`
- `scaleLineWidth`
- `shadingAxes`
- `tickObjects`
- `units`
- `verticalLine`
- `xLabel`
- `xLimSet`
- `xScale`
- `xTickLabel`

- `xTickLabelAlignment`
- `xTickLabelObjects`
- `xTickLocation`
- `xTickSet`
- `yDir`
- `yLabel`
- `yLim`
- `yLimRightSet`
- `yOffset`
- `yScaleRight`
- `yTickCount`
- `yTickCountRight`
- `yTickCountRightSet`
- `yTickRight`
- `yTickSet`
- `xTickLabelLocation`
- `xTickLabelSet`
- `xTickRotation`
- `xTickVisible`
- `yDirRight`
- `yLabelRight`
- `yLimRight`
- `yLimSet`
- `yScale`
- `yTick`
- `yTickCountObjects`
- `yTickCountRightObjects`
- `yTickRightSet`
- `yTickVisible`

- **`UIContextMenu`** ↑

Sets the `uicontextmenu` handle related to the this object

- **`alignAxes`** ↑

Set to a scalar double to align a base value for the left and right axes

- **`annotations`** ↑

A cell with annotation objects

- **`axesHandle`** ↑

A MATLAB axes object (left axes of the plot)

- **`axesHandleRight`** ↑

A MATLAB axes object (right axes of the plot)

- **`axesLabelHandle`** ↑

A MATLAB axes object (Where the axes labels are plotted)

- **`axisVisible`** ↑

If the axis of the plot should be on or off. {'on'} | 'off'

- **children** ↑

All the "nb" plotting objects of the this axes object.

- **color** ↑

{'none'} | ColorSpec ; Color of the axes back planes. Setting this property to none means that the axes is transparent and the figure color shows through. A ColorSpec is a three-element RGB vector or one of the MATLAB predefined names. Note that while the default value is none, the matlabrc.m startup file might set the axes Color to a specific color. If the 'shading' property is given this option will be overrundered.

- **colorMap** ↑

Sets the colormap used by the nb_image class if the object only plotting a image of one page. Must be given as n x 3 double or a one line char with the path to a MAT file containing the colormap. It will also set the colors used for the nb_gradedFanChart class.

For an example of a supported MAT file see:

- ...\\Examples\\Graphics\\colorMapNB.mat

- **dashType** ↑

Either char(173) (dash), char(8211) (en-dash) or char(8212) (em-dash). char(8211) is default.

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **fast** ↑

Set it to fast implementation, but where it may be a loss of functionality.

- **findAxisLimitMethod** ↑

The method used to find the limits of the axes. 1 | 2 | 3 | {4}. Default is to let MATLAB plot function find them

- **fontName** ↑

The font name used on the text of the axes.

- **fontSize** ↑

The font size used on the text of the axes. Only the x-tick and y-tick labels. See also `fontSizeX`

- **fontSizeX** ↑

The font size used on the text of the x-axis. I.e. the x-tick labels. If empty (default) the `fontSize` property is used.

- **fontUnits** ↑

```
{'points'} | 'normalized' | 'inches'  
| 'centimeters' | 'pixels'
```

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`: Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The font weight used on the text of the axes. Only the x-tick and y-tick labels

- **grid** ↑

'on' | {'off'} ; If 'on' grid lines will be added

- **gridLineStyle** ↑

'-' | '--' | ':' | '-.' | 'none'. Line style used to draw grid lines. The line style is a string consisting of a character, in quotes, specifying solid lines (-), dashed lines (--), dotted lines(:), or dash-dot lines (-.).

- **highLighted** ↑

Here will all the highlighted area handles be stored.

- **horizontalLine** ↑

Here will all of the nb_horizontalLine objects of this axes be stored. This will make it possible to automatically fit the horizontal line objects to the axes

- **imageChild** [↑](#)

The nb_image object associated with this axes. See the nb_image class.

- **language** [↑](#)

The number format on th y-axis is different for different languages. Either 'norsk' | {'english'}.

- **legend** [↑](#)

The legend of the axes, given as an nb_legend object.

- **limMode** [↑](#)

{'auto'} | 'manual' . If 'manual' the axis limits will no longer update themself automatically.

- **lineWidth** [↑](#)

Sets the line width of the axes. Default is 0.5.

- **normalized** [↑](#)

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8]. Default is 'axes'.

- **orientation** [↑](#)

Sets the orientation of the axes. Either 'vertical' (default) | 'horizontal'.

Caution : The y-axes are still the right and the left axes, and the x-axes are still the top bottom axes.

Caution : The properties xTickLabelAlignment, xTickLabelLocation, xTickLocation, and all the properties which sets how the y-axis on the right side are plotted are not supported when this property is set to 'horizontal'.

- **parent** [↑](#)

The parent, as a nb_figure object.

- **plotAxesHandle** ↑

The handle to do the plotting on. Has no axis. (When plotting on the left axes).

- **plotAxesHandleRight** ↑

The handle to do the plotting on. Has no axis. (When plotting on the right axes)

- **plotBoxAspectRatio** ↑

Set the plot box aspect ratio. Must be given as a 1x3 double [ax,ay,az]. Default is empty, which means the aspect ratio is given by the figure aspect ratio.

- **position** ↑

Position of axes. Specifies a rectangle that locates the axes within its parent container (figure or uipanel). The vector is of the form: [left bottom width height], where left and bottom define the distance from the lower-left corner of the container to the lower-left corner of the rectangle. width and height are the dimensions of the rectangle. The Units property specifies the units for all measurements.

Caution: If legend is placed outside the axes with some of the provided location property the axes will scale down accordingly. In this case this property stores the original position. See the axesHandlePosition property to get the corrected position (in normalized uints only)

- **precision** ↑

Sets the precision/format of the rounding of number on the axes.

See the precision input to the nb_num2str function. Default is [], i.e. to call num2str without additional inputs.

- **scaleFactor** ↑

Scale line width with this factor instead of the default scaling. only when scaleLineWidth is set to true.

- **scaleLineWidth** ↑

Scale line width to axes height. true or false (default).

Caution: If you change the height of the axes you need to run an update on the children of the axes to make the line adjust.

- **shading** ↑

The shading option of the axes background:

- 'grey' : Shaded grey background
- 'none' : Background color given by the color property.
- A $n \times m \times 3$ double with the background color. n is the number of horizontal pixels, m is the number of vertical pixels and 3 means the RGB colors.

- **shadingAxes** ↑

Axes of the shading

- **tickDir** ↑

{'in'} | 'out'. Direction of tick marks

- **tickObjects** ↑

Not settable

- **title** ↑

The handle to the title placed right above the axes. An nb_title handle.

- **units** ↑

Units of the position property. Default is 'normalized'.

- **update** ↑

If you want to make it possible to add children to the axes without updating the axes. This could be important for speed, when plotting more object in one axes.
Either {'on'} | 'off'.

- **verticalLine** ↑

Here will all of the nb_verticalLine objects of this axes be stored. This will make it possible to automatically fit the vertical line objects to the axes

- **visible** ↑

{'on'} | 'off'. Sets the visibility of the axes and all its children.

- **xLabel** ↑

A nb_xLabel handle (object) of the y-axis label (both or only left). Use the nb_xLabel command.

- **xLim** ↑

The limit of the x-axis. As a 1x2 double.

- **xLimSet** ↑

Not settable

- **xOffset** ↑

The offset of the x-axis tick marks.

- **xScale** ↑

Sets the scale of the x-axis. {'linear'} | 'log'.

- **xTick** ↑

The tick marks location.

- **xTickLabel** ↑

The labels of the tick marks. Either a cellstr array or a double vector

- **xTickLabelAlignment** ↑

The alignment of the x-axis tick marks labels. {'normal'} | 'middle'

- **xTickLabelLocation** ↑

The location of the x-axis tick marks labels. Either {'bottom'} | 'top' | 'baseline'. 'baseline' will only work in combination with with the xtickLocation property set to a double with the basevalue.

- **xTickLabelObjects** ↑

The text objects of the x-axis tick labels.

- **xTickLabelSet** ↑

Not settable

- **xTickLocation** ↑

The location of the x-axis tick marks. Either {'bottom'} | 'top' | double. If given as a double the tick marks will be placed at this value (where the number represent the y-axis value).

- **xTickRotation** ↑

The rotation of the x tick marks. Specify values of rotation in degrees (positive angles cause counterclockwise rotation).

- **xTickSet** ↑

Not settable

- **xTickVisible** ↑

Sets the visibility of the x-axis tick marks. 1 visible and 0 invisible.

- **yDir** ↑

{'normal'} | 'reverse'. The direction of the y-axis (both or only left)

- **yDirRight** ↑

{'normal'} | 'reverse'. The direction of the y-axis (only right)

- **yLabel** ↑

The nb_yLabel handle (object) of the y-axis labels (only left).

- **yLabelRight** ↑

The nb_yLabel handle (object) of the y-axis labels (only right).

- **yLim** ↑

The limit of the y-axis (both or only left). Default limits is [0 1], on both axes.

- **yLimRight** ↑

The limit of the y-axis (only right). As long as there is not plotted any nb_* plotting objects on this axes this property will not be used, but of course if you set it.

- **yLimRightSet** ↑

Not settable

- **yLimSet** ↑

Not settable

- **yOffset** ↑

The offset of the y-axis tick marks

- **yScale** ↑

Sets the scale of the left y-axis (or both if nothing is plotted on the right axes). {'linear'} | 'log'.

- **yScaleRight** ↑

Sets the scale of the right y-axis. {'linear'} | 'log'.

- **yTick** ↑

The tick marks location (both or only left)

- **yTickLabel** ↑

The labels of the tick marks (both or only left). Either a cellstr array or a double vector

- **yTickLabelObjects** ↑

The text objects of the y-axis tick labels (left)

- **yTickLabelRight** ↑

The labels of the tick marks (only right). Either a cellstr array or a double vector

- **yTickLabelRightObjects** ↑

The text objects of the y-axis tick labels (right)

- **yTickLabelRightSet** ↑

Not settable

- **yTickLabelSet** ↑

Not settable

- **yTickRight** ↑

The tick marks location (only right)

- **yTickRightSet** ↑

Not settable

- **yTickSet** ↑

Not settable

- **yTickVisible** ↑

Sets the visibility of the y-axis tick marks. 1 visible and 0 invisible.

Methods:

- [get](#)
 - [set](#)
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_axes

Input:

- obj : An object of class nb_axes
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('color');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_axes. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_axes
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_axes with the given properties reset

Examples:

```
obj = obj.set('position',[0.1 0.1 0.8 0.8],'xLim',[0,1],...  
'yLim',[0.1]);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_bar

Go to: [Properties](#) | [Methods](#)

```
obj = nb_bar(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making bar plots with or without shaded bars.

```
nb_bar(yData)  
nb_bar(xData,yData)  
nb_bar(xData,yData,'propertyName',PropertyValue,...)  
handle = nb_bar(xData,yData,'propertyName',PropertyValue,...)
```

The handle have set and get methods as all MATLAB graph handles.

Caution: All the children of this object is of class nb_patch

Constructor:

```
obj = nb_bar(xData,yData,varargin)
```

Input:

- xData : The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)
- yData : The yData of the plot. The columns are counted as separate variables
- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_bar

Examples:

```
obj = nb_bar([1,2]); % Provide only y-axis data
obj = nb_bar([1,2],[1,2]);
obj = nb_bar([1,2], 'propertyName', PropertyValue, ...)
obj = nb_bar([1,2],[1,2], 'propertyName', PropertyValue, ...)
```

See also:

bar

Written by Kenneth Sæterhagen Paulsen

Properties:

- alpha1
- alpha2
- barWidth
- baseValue
- baseline
- blend
- cData
- children
- deleteOption
- direction
- edgeColor
- fast
- legendInfo
- lineObj
- lineWidth
- parent
- scale
- shadeColor
- shaded
- side
- style
- sumTo
- visible
- xData
- yData

- alpha1 ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- alpha2 ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- barWidth ↑

Width of the bars. Must be a scalar.

- baseValue ↑

The base value of the plot. Must be a scalar or a vector with size size(yData,1) x 1.

- **baseline** ↑

A nb_horizontalLine handle (object). Use the set and get methods of this handle to change the baseline properties.

Caution: If baseValue is a vector this property is a nb_line object.

- **blend** ↑

When shaded is used, and this option is set to true, it will do alpha blending with shadeColor instead of shading.

- **cData** ↑

Color of the plotted data. A matrix; size(yData,2) x 3 (RGB color). Or a cellstr with the color names. (With size 1 x size(yData,2))

- **children** ↑

All the handles of the bar plot.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **direction** ↑

The direction of the shading; {'north'} | 'south' | 'east' | 'west'

- **edgeColor** ↑

The color of the edge of the bars. In RGB colors or a string with the color names. Can also be 'none' ; no edgeLine or 'same' ; same color as the base color for each bar. Can also be a cellstr with either the RGB colors or color names with size 1 x size(yData,2).

- **fast** ↑

Is the fast or general algorithm beeing used.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineObj** ↑

A nb_line handle representing the sum of the stacked bars when 'style' is set to 'dec'. As a line handle.

- **lineStyle** ↑

Sets the line style(s) of the edge of the bars.
Either a string or a cellstr (which must have
the size 1 x size(yData,2)).

- **lineWidth** ↑

The line width of the edge of the bars. Must be a scalar.

- **parent** ↑

The parent axes you wan to plot on, if not given it will
be plotted in a new nb_axes handle. Either a axes or a
nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **scale** ↑

Sets the tightness of the bars plotted. (Stacked)

- **shadeColor** ↑

The color (RGB) the shaded bar plots are
interpolated with. Either a 1x3 double with the
RGB colors or a string with the color name.
Default is [1,1,1]

- **shaded** ↑

Index of which data should be shaded. A matrix with size;
size(yData,1) x 1 or 1 x size(yData,1) or size(yData,1) x
size(yData,2)

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the
parent is of class nb_axes. (Which is the default)

- **style** ↑

'stacked' | {'grouped'} | 'dec'. When 'dec' is given the
bars will be stacked but there will also be plotted a
line with the sum. (It is not possible to use the sumTo
option at the same time)

- **sumTo** ↑

If given the sum of the bars for each period will sum up to this number. (only when style is set to 'stacked')
E.g. if set 100, each bar for each variable will be given as percentage share of the total sum.

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **xData** ↑

The xData of the plotted data. Must be of size;
size(yData,1) x 1 or 1 x size(yData,1)

- **yData** ↑

The yData of the plot. The columns are counted as separate variables.

Methods:

- findClosestNiceNumber
- findLimitsAlgo
- get
- getDefaultColors
- interpretColor
- set

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                 scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_bar

Input:

- obj : An object of class nb_bar
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors ↑**

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ!terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_bar. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_bar
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_bar with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_barAnnotation

Go to: [Properties](#) | [Methods](#)

`nb_textAnnotation, nb_annotation, handle`

Description:

A class for plotting annotation on bar plots.

Constructor:

```
obj = nb_barAnnotation(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_barAnnotation

Examples:

See also:

`nb_textAnnotation, nb_annotation, handle, nb_graph_ts`
`nb_graph_cs`

Written by Kenneth Sætherhagen Paulsen

Properties:

- `children`
- `color`
- `copyOption`
- `decimals`
- `deleteOption`
- `fontName`
- `fontSize`
- `fontUnits`
- `fontWeight`
- `force`
- `language`
- `listeners`
- `location`
- `normalized`
- `parent`
- `pointer`
- `rotation`
- `selected`
- `space`
- `string`
- `type`

- **`children`** ↑

The children of the handle.

Inherited from superclass `NB_ANNOTATION`

- **`color`** ↑

Color of the text of the annotation. Either a string with the color name or a 1x3 double with the RGB colors.

- **`copyOption`** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass `NB_ANNOTATION`

- **`decimals`** ↑

The number of decimals wanted on the text. Only when type is set to 'default'.

- **`deleteOption`** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass `NB_ANNOTATION`

- **`fontName`** ↑

Name of the font to use. Must be a string. Default is 'arial'.

Inherited from superclass `NB_TEXTANNOTATION`

- **`fontSize`** ↑

Sets the font size in the units given by the font units property. Must be a scalar. Default is 12.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontUnits** ↑
{'points'} | 'normalized' | 'inches' | 'centimeters' |
'pixels'

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`:
Absolute units. 1 point = 1/72 inch.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontWeight** ↑

Weight of text characters. {'normal'} | 'bold' | 'light'
| 'demi'

Inherited from superclass `NB_TEXTANNOTATION`

- **force** ↑

Force the number of decimals. I.e. even integers get zeros as decimals.

- **language** ↑

Sets the language of the text. I.e. how decimal sign should be plotted. Only when the input to the `plot` function is of class `nb_bar`. {'english'} | 'norwegian'.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass `NB_ANNOTATION`

- **location** ↑

Sets the location of the annotation.
{'on'} : Place it on the bars
'top' ; Place it on the top of the bars

- **normalized** ↑

Indicate if the font when `fontUnits` is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

Inherited from superclass `NB_TEXTANNOTATION`

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **rotation** ↑

The rotation of the text. As a double in degrees.

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **space** ↑

Sets the space between the text and the bar when location is set to top. The scale factor is given as the share of the height of the y-axis. Only an option when location is set to 'top'.

- **string** ↑

Set the text of the bars. Must be a cellstr with same size as the plotted data. Where the rows will be the time periods/case dimension and the columns will be the given variables. I.e. in line with how the data is orginized in nb_ts and the nb_cs classes

E.g: {'2,0','3,0','4,0','5,0'}

- **type** ↑

Sets the how the text on the bars should be created:

- {'default'} : The text represent the contribution
- 'manual' : The text is set manually by the user. See property string (When the string property is set this will be the default value of this property)

Methods:

- **set**

► **set** ↑

set(obj,varargin)

Description:

Sets the properties of an nb_barAnnotation object

Input:

- obj : An object of class nb_drawLine
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output

- obj : An object of class nb_barAnnotation with the wanted properties set.

Examples:

Written by Kenneth Sæterhagen Paulsen

■ nb_candle

Go to: [Properties](#) | [Methods](#)

```
obj = nb_candle(xData,high,low,open,close,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making bar plots with or without shaded bars.

```
nb_candle(xData,high,low,open,close,...  
          'propertyName',PropertyValue,...)  
handle = nb_candle(xData,high,low,open,close,...  
                    'propertyName',PropertyValue,...)
```

The handle have set and get methods as all MATLAB graph handles.

Caution: All the children of this object is of class nb_patch and nb_line

Constructor:

```
obj = nb_candle(xData,high,low,open,close,varargin)
```

Input:

- xData : The xData of the plotted data. Must be of size; size(high,1) x 1 or 1 x size(high,1)
- high : The highest observation. Must be a double

vector. Can be given as a empty double.

- low : The lowest observation. Must be a double vector.
Can be given as a empty double.
- open : The highest value of the plotted patch. Must be
a double vector. Can be given as a empty double.
- close : The lowest value of the plotted patch. Must be
a double vector. Can be given as a empty double.
- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_candle

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData
- close
- edgeColor
- indicator
- indicatorLineWidth
- lineWidthStyle
- marker
- shadeColor
- visible
- candleWidth
- deleteOption
- high
- indicatorColor
- indicatorWidth
- lineWidth
- open
- shaded
- xData
- children
- direction
- hLineWidth
- indicatorLineStyle
- legendInfo
- low
- parent
- side

- cData ↑

Color of the plotted patches. A matrix;
1 x 3 (RGB color) or a string with the color
name.

- candleWidth ↑

Width of the candles. Must be a scalar.

- children ↑

All the handles of the bar plot.

- close ↑

The lowest values of the plotted patches. Must be double vector. Must match the open, high and low properties, if they are not empty.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **direction** ↑

The direction of the shading; {'north'} | 'south' | 'east' | 'west'

- **edgeColor** ↑

The color of the edge of the bars. In RGB colors or a string with the color names. Can also be 'none' ; no edgeLine or 'same' ; same color as the base color for each bar. Can also be a cellstr with either the RGB colors or color names with size 1 x size(yData,2).

- **high** ↑

The highest values of the plotted candles. Must be double vector. Must match the close, open and low properties, if they are not empty.

- **hlLineWidth** ↑

The line width of the high and low lines. As a scalar. Default is 2.

- **indicator** ↑

The value for where to plot a vertical line. Can be given as an empty double. Must match the close, open, high and low properties, if they are not empty.

- **indicatorColor** ↑

The color of the indicator line. As a 1 x 3 double or a string with the color name. Default is [0,0,0].

- **indicatorLineStyle** ↑

Sets the indicator line style. Default is '-'.

- **indicatorLineWidth** ↑

The line width of the indicator line. As a scalar. Default is 4.

- **indicatorWidth** ↑

The width of the indicator. The default is to use the same width as provided by the candleWidth property. I.e. when this property is set to [].

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

Sets the line style(s) of the edge of the bars. Either a string or a cellstr (which must have the size 1 x size(yData,2)).

- **lineWidth** ↑

The line width of the edge of the bars. Must be a scalar.

- **low** ↑

The lowest values of the plotted candles, Must be double vector. Must match the close, high and open properties, if they are not empty.

- **marker** ↑

Sets the marker of the indicator line. Default is 'none'.

- **open** ↑

The highest values of the plotted patches. Must be double vector. Must match the close, high and low properties, if they are not empty.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **shadeColor** ↑

```
The color (RGB) the shaded bar plots are  
interpolated with. Either a 1x3 double with the  
RGB colors or a string with the color name.  
Default is [1,1,1]
```

- **shaded** ↑

```
Index of which data should be shaded. A vector with size;  
size(open,1) x 1 or 1 x size(open,1).
```

- **side** ↑

```
{'left'} | 'right' ; Which axes to plot on. Only if the  
parent is of class nb_axes. (Which is the default)
```

- **visible** ↑

```
Sets the visibility of the plotted lines. {'on'} | 'off'
```

- **xData** ↑

```
The xData of the plotted data. Must be of size;  
size(high,1) x 1 or 1 x size(high,1)
```

Methods:

- [findClosestNiceNumber](#)
- [findLimitsAlgo](#)
- [get](#)
- [getDefaultsColors](#)
- [interpretColor](#)
- [set](#)

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number,...  
scaleNumber,up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits
when findAxisMethod == 3 is used.

Input:

- **number** : The given number to find a "nice" number close to.
- **scaleNumber** : The scale factor used. I.e. the precision of the algorithm.
- **up** : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► findLimitsAlgo ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_candle

Input:

- obj : An object of class nb_candle
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_candle. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_candle
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_candle with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ **nb_colorbar**

Go to: [Properties](#) | [Methods](#)

nb_annotation, handle

Description:

A class for adding color bar to a plot (axes).

Constructor:

```
obj = nb_colorbar(varargin)
```

Input:

Optional input ('propertyName', PropertyValue) :

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_colorbar

See also:

[nb_annotation](#), [handle](#), [nb_graph_cs](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [ID](#)
- [fontName](#)
- [language](#)
- [pointer](#)
- [tickLabelsSet](#)
- [children](#)
- [fontSize](#)
- [listeners](#)
- [selected](#)
- [copyOption](#)
- [fontUnits](#)
- [location](#)
- [space](#)
- [deleteOption](#)
- [fontWeight](#)
- [location](#)
- [strip](#)
- [direction](#)
- [invert](#)
- [parent](#)
- [tickLabels](#)
- [ticks](#)
- [ticksSet](#)

- [ID](#) ↑

`nb_colorbar.ID` is a property.

- [children](#) ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- [copyOption](#) ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- [deleteOption](#) ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- [direction](#) ↑

Direction of the color bar. 'normal' or 'reverse'.

- [fontName](#) ↑

Name of the font to use. Must be a string. Default is 'arial'.

Inherited from superclass NB_TEXTANNOTATION

- [fontSize](#) ↑

Sets the font size in the units given by the font units property. Must be a scalar. Default is 12.

Inherited from superclass NB_TEXTANNOTATION

- **fontUnits** ↑
{'points'} | 'normalized' | 'inches' | 'centimeters' |
'pixels'

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`:
Absolute units. 1 point = 1/72 inch.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontWeight** ↑

Weight of text characters. {'normal'} | 'bold' | 'light'
| 'demi'

Inherited from superclass `NB_TEXTANNOTATION`

- **invert** ↑

Invert color map when applied to the color bar. `true` or `false`. Default is `false`.

- **language** ↑

Sets the language of the text. I.e. how decimal sign should be plotted. {'english'} | 'norwegian'.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass `NB_ANNOTATION`

- **location** ↑

The location of the color bar. See the MATLAB function `colorbar` for more on the values to choose from. Default is '`eastOutside`'.

- **normalized** ↑

Indicate if the font when `fontUnits` is set to 'normalized' should be normalized to the axes ('`axes`') or to the figure ('`figure`'), i.e. the default axes position [0.1 0.1 0.8 0.8].

Inherited from superclass `NB_TEXTANNOTATION`

- **parent** ↑

The parent as an `nb_axes` object

Inherited from superclass `NB_ANNOTATION`

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **space** ↑

Extra space added between color bar and the assosiated axes.
Positive scalar number.

- **strip** ↑

Strip colors from the color map of the parent to be in the
color bare. Default is to strip [1,1,1] (white). Must be a
N x 3 double.

- **tickLabels** ↑

Tick mark labels, specified as a cell array of character vectors,
a string array, a numeric array, a character vector, or a
categorical array. By default, the colorbar labels the tick
marks with numeric values taken from the data of the imageChild
of the nb_axes the colorbar is attached to. If this property is
empty (no image added to the axes), a default [0,1] scale is
given.

- **tickLabelsSet** ↑

nb_colorbar/tickLabelsSet is a property.

- **ticks** ↑

Tick mark locations, specified as a vector of monotonically
increasing numeric values. The values do not need to be equally
spaced. If you do not want tick marks displayed, then set the
property to the empty vector, [].

- **ticksSet** ↑

nb_colorbar/ticksSet is a property.

Methods:

- [set](#)

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_arrow. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_arrow
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_arrow with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_curlyBrace

Go to: [Properties](#) | [Methods](#)

```
nb_movableAnnotation, nb_lineAnnotation, nb_annotation, handle
```

Description:

A class for adding a curly brace to the plot.

The code for plotting the curly brace is gotten from the drawbrace functionn made by Pål Nærli Sævik.

Constructor:

```
obj = nb_curlyBrace(varargin)
```

Input:

Optional input ('propertyName', PropertyValue) :

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_curlyBrace

See also:

[nb_annotation](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [cData](#)
- [children](#)
- [clipping](#)
- [copyOption](#)
- [deleteOption](#)
- [lineStyle](#)
- [lineWidth](#)
- [listeners](#)
- [marker](#)
- [markerEdgeColor](#)
- [markerFaceColor](#)
- [markerSize](#)
- [parent](#)
- [pointer](#)
- [selected](#)
- [side](#)
- [units](#)
- [width](#)
- [xData](#)
- [yData](#)

- **cData** ↑

The color data of the line plotted. Must be of size;
1 x 3. With the RGB colors. Or a string with the name of
the color. See the method `interpretColor` of the
`nb_plotHandle` class for more on the supported color names.

- **children** ↑

The children of the handle.

Inherited from superclass `NB_ANNOTATION`

- **clipping** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the
axes plot box by default. If you set Clipping to off,
lines are displayed outside the axes plot box.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu
of annotation object. Default is false.

Inherited from superclass `NB_ANNOTATION`

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be
deleted and all that is plotted by this object are
removed from the figure it is plotted on. If on the other
hand 'only' is given, it will just delete the object

Inherited from superclass `NB_ANNOTATION`

- **lineStyle** ↑

The line style of the plotted data. Must be a string.
{'-' } | '--' | ':' | '-.' | 'none'. '---' is not
supported.

- **lineWidth** ↑

The line width of the plotted data. As an integer.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **marker** ↑

The markers of the lines. Must be a string. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerSize** ↑

Size of the markers. Must be a integer. Default is 9.

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

The axes to plot against. 'left' or 'right'. 'left' is default.

- **units** ↑

The coordinates system to plot in. Either {'data'} : xy-coordinates | 'normalized' : figure coordinates.

- **width** ↑

Width of the brace. Try and fale method must be used!

- **xData** ↑

The x-axis data. As a 1x2 double.

- **yData** ↑

The y-axis data. As a 1x2 double.

Methods:

- **set**

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_drawLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- `obj` : An object of class nb_drawLine
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class nb_drawLine with the given properties reset

Examples:

`obj.set('propertyName',PropertyValue,...);`

Written by Kenneth Sæterhagen Paulsen

■ **nb_donut**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_donut(yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making donut charts in 2D.

This class will make the donut chart automatically fit the axes of the plot.

I recommend to use the nb_* classes for the legend, titles and so on.

Constructor:

```
obj = nb_donut(yData,varargin)
```

Input:

- yData : The data to plot. min(size(yData)) must be 1.

Optional input:

- varargin : 'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_pie (handle).

Examples:

```
nb_donut(yData)
nb_donut(yData,'propertyName',PropertyValue,...)
handle = nb_donut(yData,'propertyName',PropertyValue,...)
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- axisVisible
- bite
- cData
- children
- deleteOption
- edgeColor
- explode
- fontColor
- fontName
- fontSize
- fontUnits
- fontWeight
- innerRadius
- labelHandles
- labels
- labelsExtension
- legendInfo
- lineStyle
- lineWidth
- location
- noLabels
- origoPosition
- parent
- radius
- visible
- yData

- **axisVisible** ↑

Make the axes box visible ('on') or not ('off'). 'on' is default.

- **bite** ↑

A logical array with ones for the slice to bite. I.e. [0,0,1,0]; Must be of size; 1 x nSlices.

- **cData** ↑

The colors of the plotted data. Must be of size nSlices x 3. (Double with the RGB colors), or a cellstr with the color names. Must be of size 1 x nSlices. See the method nb_plotHandle.interpretColor for more on the supported color names.

- **children** ↑

The handles of all the children of the plot.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **edgeColor** ↑

The color of the edge of the pies. Must be of size; 1 x 3. (Double with the RGB colors), or a one line char with the color name. See the method nb_plotHandle.interpretColor for more on the supported color names.

- **explode** ↑

A logical array with ones for the pies to explode. I.e. [0,0,1,0]; Must be of size 1 x nSlices.

- **fontColor** ↑

Color of labels. Must be double with size 1 x 3 (RGB colors), or a string with the name of the color. See the method nb_plotHandle.interpretColor for more on the supported color names.

- **fontName** ↑

Name of the font used. Default is 'arial'.

- **fontSize** ↑

Font size of the text on the plot. Default is 12.

- **fontUnits** ↑

```
{'points'} | 'normalized' | 'inches' |  
'centimeters' | 'pixels'
```

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

Font weight of the text on the plot. Default is 'normal'.

- **innerRadius** ↑

Sets the relative length of the inner circle of the donut. -0.5 mean that the donut has an inner circle with the same radius as half the outer circle of the donut.

- **labelHandles** ↑

Handles to the labels of the pie plot. As MATLAB text objects.

- **labels** ↑

The labels of the pies, must be a cellstr with `size`; `length(yData)`. If not given the labels will be the shares of each pie in percent.

- **labelsExtension** ↑

If you not give the labels you can set the extension of the labels. Default is '%'. The extension will be added to the value of the data. If '%' is given the data will be in relative shares in percent.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend.

- **lineStyle** ↑

Line style of the edge.

- **lineWidth** ↑

Line width of the edge.

- **location** ↑

{'west'} | 'east' | 'center'. The location of the pie chart in the current axes.

- **noLabels** ↑

Force there to be no labels of the pies. If yData gives more donuts noLabels will be forced to 1 (true).

- **origoPosition** ↑

Set the origo position. Default is to use the location property. Where 'west' is [-0.5,0], 'center' is [0,0] and 'east' is [0.5,0]. Must either be empty or a 1x2 double.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **radius** ↑

Set the radius of the nb_donut. Default is 1.

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **yData** ↑

The data to plot. As a nDonuts x nSlices.

Caution: If given as N x 1 it is interpreted as 1 x nSlices.

Methods:

- [findClosestNiceNumber](#)
- [findLimitsAlgo](#)
- [get](#)
- [getDefaultValue](#)
- [interpretColor](#)
- [set](#)

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number,...  
scaleNumber,up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► findLimitsAlgo ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)

```
> 3: Uses the submethod findClosestNiceNumber of  
      the class to decide on the limits. (Also this  
      method have can have problems with data which  
      is not symmetric around the zero line.)  
> 4 : Uses the MATLAB algorithm for finding the  
      axis limits.
```

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_donut

Input:

- obj : An object of class nb_donut
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► interpretColor ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÃ,d','r'  
> 'green','grÃ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÃ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÃ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÃ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colormames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► set ↑

set(obj,varargin)

Description:

Sets the properties of an object of class nb_donut. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_donut
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_donut with the given properties reset

Examples:

obj.set('propertyName',PropertyValue,...);

Written by Kenneth Sæterhagen Paulsen

■ nb_drawCircle

Go to: [Properties](#) | [Methods](#)

[nb_lineAnnotation](#), [nb_movableAnnotation](#), [nb_annotation](#), [handle](#)

Description:

A class for adding a arbitrary circle to a plot using the patch handle class.

Constructor:

```
obj = nb_drawCircle(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class.

Output

- obj : An object of class nb_drawPatch

See also:

[nb_annotation](#), [handle](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- children
- clipping
- copyOption
- deleteOption
- edgeColor
- faceColor
- lineStyle
- lineWidth
- listeners
- parent
- pointer
- rData
- selected
- side
- units
- xData
- yData

- **children** [↑](#)

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **clipping** [↑](#)

'on' or 'off'. Clipping to axes rectangle. When Clipping is on, MATLAB does not display any portion of the patch outside the axes rectangle.

- **copyOption** [↑](#)

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **deleteOption** [↑](#)

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **edgeColor** ↑

The color of the edge of the rectangle. In RGB colors | 'none' : No edgeLine | 'same' : Same color as the face color | a string with the color name.

- **faceColor** ↑

The RGB color of the face of the rectangle.

Must be a 1 x 3 double with the RGB colors or a string with the color name. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- **lineStyle** ↑

The line style of the edge of the patch.

- **lineWidth** ↑

The line width of the edge of the patch.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **rData** ↑

Radius at the center of the circle in the y-direction.
Default is 1.

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **units** ↑

The coordinates system to plot in. Either {'data'} :
xy-coordinates | 'normalized' : figure coordinates.

- **xData** ↑

The x-axis data. As a 1x2 double. E.g: [x_begin, x_end].

- **yData** ↑

The y-axis data. As a 1x2 double. E.g: [y_begin, y_end].

Methods:

- **set**

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_drawLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- `obj` : An object of class nb_drawLine
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class nb_drawLine with the given properties reset

Examples:

`obj.set('propertyName',propertyValue,...);`

Written by Kenneth Sæterhagen Paulsen

■ **nb_drawLine**

Go to: [Properties](#) | [Methods](#)

```
nb_lineAnnotation, nb_annotation, handle
```

Description:

A class for adding a arbitrary line to a plot.

Constructor:

```
obj = nb_drawLine(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_drawLine

Examples:

See also:

[nb_annotation](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [cData](#)
- [children](#)
- [clipping](#)
- [copyOption](#)
- [deleteOption](#)
- [lineStyle](#)
- [lineWidth](#)
- [listeners](#)
- [marker](#)
- [markerEdgeColor](#)
- [markerFaceColor](#)
- [markerSize](#)
- [parent](#)
- [pointer](#)
- [selected](#)
- [side](#)
- [units](#)
- [xData](#)
- [yData](#)

- **cData** ↑

The color data of the line plotted. Must be of size;
1 x 3. With the RGB colors. Or a string with the name of
the color. See the method `interpretColor` of the
`nb_plotHandle` class for more on the supported color names.

- **children** ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **clipping** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the
axes plot box by default. If you set Clipping to off,
lines are displayed outside the axes plot box.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **lineStyle** ↑

The line style of the plotted data. Must be a string. {'-' } | '--' | ':' | '-' | 'none'. '---' is not supported.

- **lineWidth** ↑

The line width of the plotted data. As an integer.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **marker** ↑

The markers of the lines. Must be a string. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerSize** ↑

Size of the markers. Must be a integer. Default is 9.

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

The axes to plot against. 'left' or 'right'. 'left' is default.

- **units** ↑

The coordinates system to plot in. Either {'data'} : xy-coordinates | 'normalized' : figure coordinates.

- **xData** ↑

The x-axis data. As a double.

- **yData** ↑

The y-axis data. As a double.

Methods:

- **set**

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_drawLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- **obj** : An object of class nb_drawLine
- **varargin** : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_drawLine with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_drawPatch

Go to: [Properties](#) | [Methods](#)

nb_lineAnnotation, nb_movableAnnotation, nb_annotation, handle

Description:

A class for adding a arbitrary patch to a plot using the rectangle handle class.

Constructor:

```
obj = nb_drawPatch(varargin)
```

Input:

Optional input ('propertyName', PropertyValue) :

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_drawPatch

Examples:

See also:

[nb_annotation](#), [handle](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- children
- clipping
- copyOption
- curvature
- deleteOption
- edgeColor
- faceColor
- lineStyle
- lineWidth
- listeners
- parent
- pointer
- position
- selected
- side
- units

- **children** ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **clipping** ↑

'on' or 'off'. Clipping to axes rectangle. When Clipping is on, MATLAB does not display any portion of the patch outside the axes rectangle.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **curvature** ↑

one- or two-element vector [x,y]. Default is [0,0]

Amount of horizontal and vertical curvature. Specifies the curvature of the rectangle sides, which enables the shape of the rectangle to vary from rectangular to ellipsoidal. The horizontal curvature x is the fraction of width of the rectangle that is curved along the top and bottom edges. The vertical curvature y is the fraction of the height of the rectangle that is curved along the left and right edges.

The values of x and y can range from 0 (no curvature) to 1 (maximum curvature). A value of [0,0] creates a rectangle with square sides. A value of [1,1] creates an ellipse. If you specify only one value for Curvature, then the same length (in axes data units) is curved along both horizontal and vertical sides. The amount of curvature is determined by the shorter dimension

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **edgeColor** ↑

The color of the edge of the rectangle. In RGB colors | 'none' : No edgeLine | 'same' : Same color as the face color | a string with the color name.

- **faceColor** ↑

The RGB color of the face of the rectangle.

Must be a 1 x 3 double with the RGB colors or a string with the color name. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- **lineStyle** ↑

The line style of the edge of the patch.

- **lineWidth** ↑

The line width of the edge of the patch.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **position** ↑

The position of the MATLAB rectangle object.

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **units** ↑

The coordinates system to plot in. Either {'data'} :
xy-coordinates | 'normalized' : figure coordinates.

Methods:

- **set**

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_drawLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_drawLine
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_drawLine with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_fanChart

Go to: [Properties](#) | [Methods](#)

```
obj = nb_fanChart(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for plotting fan chart.

Constructor:

```
nb_fanChart(yData)
nb_fanChart(xData,yData)
nb_fanChart(xData,yData,'PropertyName',PropertyValue,...)
handle = nb_fanChart(xData,yData,'PropertyName',...
                      PropertyValue,...)
```

Input:

- xData : The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)

```
- yData      : The yData of the plot. The columns are counted  
               as different simulations. And on the basis of this  
               data the confidence intervals are calculated.
```

```
- varargin : ...,'propertyName',PropertyValue,...
```

Output

```
- obj       : An object of class nb_fanChart
```

Examples:

```
obj = nb_fanChart(rand(2,100)); % Provide only y-axis data  
obj = nb_fanChart([1,2],rand(2,100));  
obj = nb_fanChart(rand(2,100),'propertyName',PropertyValue,...)  
obj = nb_fanChart([1,2],rand(2,100),'propertyName',...  
                  PropertyValue,...)
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- alpha
- cData
- central
- children
- deleteOption
- legendInfo
- lineWidth
- method
- parent
- percentiles
- side
- visible
- xData
- yData

- **alpha** ↑

Transparency of fan chart. A double between 0 and 1. 0 fully transparent, 1 opaque.

- **cData** ↑

Either double with the RGB color specification for each percentile, or a string with the color specification to use ({'nb'} | 'red' | 'green' | 'yellow' | '')

- **central** ↑

An nb_line object with the mean, if you set this as a string the mean will not be plotted

- **children** ↑

The children of the plot, as MATLAB patch handles.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be deleted and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **legendInfo** ↑

{'off'} : The mean line will not be included in the legend. 'on' : included it in the legend

- **lineWidth** ↑

The line width of center of the fan. See also property center.

- **method** ↑

{'percentiles'} | 'hdi'; Which type of fan chart is going to be made. 'percentiles' are produced by using percentiles, while 'hdi' use highest density intervals

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **percentiles** ↑

The percentiles as 1 x numberOfPercentiles double. E.g:
[.3,.5,.7,.9] (Which is default)

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **visible** ↑

Set the visibility of the fan chart.

- **xData** ↑

The x-axis data. Should be a vector.

- **yData** ↑

Each column of the yData property is counted as one simulation. And on the basis of this data the confidence intervals are calculated.

If this input has more than one page, it will count each page as a seperate variable, and each variable will get its own fan around the central tendency. In this case percentiles property must be set to a 1x1 double.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► [findClosestNiceNumber](#) ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► [findLimitsAlgo](#) ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_fanChart

Input:

- obj : An object of class nb_fanChart
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('central');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'
```

```
> 'grey','grÃ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÃ¥','hb'  
> 'deep blue','mÃ¸rk blÃ¥','db'  
> 'water blue','vannblÃ¥','vb'  
> 'cool grey','kald grÃ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_fanChart. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_fanChart
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_fanChart with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_fanLegend

Go to: [Properties](#) | [Methods](#)

```
obj = nb_fanLegend(parent,string,varargin)
```

Description:

This is a class for creating MPR looking legends for the fan charts

Constructor:

```
obj = nb_fanLegend(parent,string,varargin)
```

Input:

- parent : An object of class nb_axes. One of its children must be of class nb_fanChart.
- string : Sets the text over the colored boxes of the MPR looking fan legend, you must give them as a cellstr. Must match the number of percentiles of the graph created by the nb_fanChart object. E.g. {'30%','50%','70%','90%'}). If empty the percentiles of the underling nb_fanChart object will be used.
- varargin : Property name, property value combinations.

Output

- obj : An object of class nb_fanLegend

Examples:

```
fan = nb_fanChart([ones(20,100)*0.5;rand(10,100)],...  
'cData','grey');  
obj = nb_fanLegend(fan.parent,{})
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|------------|-------------|----------------|---------------|------------|
| • children | • colors | • deleteOption | • extent | • fontName |
| • fontSize | • fontUnits | • fontWeight | • interpreter | • location |
| • parent | • string | • visible | | |

- **children** ↑

The children of the fan legend. As MATLAB handles.

- **colors** ↑

Colors of the fan layers. As a M X 3 double with the RGB colors. Should not be set.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **extent** ↑

The extent of the fan legend. A 1x2 double with the lower left point.

- **fontName** ↑

The font name to use. As a string. 'arial' is default.

- **fontSize** ↑

The size of the font. As a scalar. Default is 14.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The weight of the font, {'bold'} | 'normal' | 'light'.

- **interpreter** ↑

{'none'} | 'tex' | 'latex'

- **location** ↑

Sets the location of the fan legend. Must be a string or a 1 x 2 double.

Different location supported:

- 'Best' : (same as 'North').)
- 'NorthWest'
- 'North'
- 'NorthEast'
- 'SouthEast'
- 'South'
- 'SouthWest'
- 'outside' : Places the legend outside the axes, one the right side. (If you create subplots, this is the only option which will look good.)

You can also give a 1 x 2 double vector with the location of where to place the legend. First column is the x-axis location and the second column is the y-axis location. Both must be between 0 and 1.

- **parent** ↑

The parent, as a nb_axes handle (object)

- **string** ↑

Text above the squared colorboxes of the legend. As a cellstr. Must match the number of percentiles of the underlying nb_fanChart object. E.g. {'30%', '50%', '70%', '90%'}). If empty the percentiles of the underling nb_fanChart object will be used.

- **visible** ↑

{'on'} | 'off'.

Methods:

- **get**
- **set**

► **get** ↑

```
propertyValue = get(obj, propertyName)
```

Description:

Get a property of an object of class nb_title

Input:

- obj : An object of class nb_title
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_fanLegend. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_fanLegend
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_fanLegend with the given properties reset

Examples:

```
obj.set('propertyName',propertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_figure

Go to: [Properties](#) | [Methods](#)

```
obj = nb_figure(varargin)
```

Superclasses:

handle

Description:

This is a object representing a MATLAB figure. This class extends the MATLAB figure class.

Through the figureHandle property you have all the MATALB functionalities, but you can also store nb_axes objects (not the MATLAB class axes), which again can store nb_bar, nb_pie, nb_radar, nb_plot, nb_area, nb_patch, nb_scatter and nb_plotComb objects.

This class makes it also possible to add a title for the whole figure which will be placed above all the children (axes including text objects). And it makes it possible to add a footer to the figure below all the children (axes including text objects) of the figure. Set the properties figureTitle and footer respectively.

These classes makes it more easy to create nice graphics.

Constructor:

```
obj = nb_figure(varargin)
```

Input:

- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_figure

Examples:

```
nb_figure  
obj = nb_figure()  
obj = nb_figure('propertyName', PropertyValue, ...)
```

See also:

[figure](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- children
- deleteOption
- figureHandle
- figureTitle
- footer
- userData

- **children** ↑

The children of the figure. Must be of class nb_axes.

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be deleted and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **figureHandle** ↑

A MATLAB figure handle.

- **figureTitle** ↑

The title of the figure. Placed above all the figure axes as an nb_figureTitle object.

- **footer** ↑

The footer placed below all the figure axes as an nb_footer object.

- **userData** ↑

User data

Methods:

- get
- getInnerExtent
- set

► **get** ↑

propertyValue = get(obj,propertyName)

Description:

Get a property of an object of class nb_figure

Input:

- obj : An object of class nb_figure
- propertyName : A string with the propertyName

Output:

- `propertyValue` : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getInnerExtent** ↑

```
extent = getInnerExtent(obj,units)
```

Description:

Get inner extent of all children in a nb_figure object.

Written by Kenneth Sætherhagen Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_figure. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- `obj` : An object of class nb_figure
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class nb_figure with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_figureTitle

Go to: [Properties](#) | [Methods](#)

```
obj = nb_figureTitle(string,varargin)
```

Superclasses:

handle

Description:

This is a class for making figure title on an nb_figure object.
Placed above all elements of the figure

Constructor:

```
obj = nb_figureTitle(string,varargin)
```

Input:

- string : The figure title, as a string or a char with
the text. If it is a char, each row will be
added as a separate lines.

- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_figureTitle

Examples:

```
nb_figureTitle('A figure title')
ft = nb_figureTitle('A figure title')
ft = nb_figureTitle(char('A figure title','New line'))
ft = nb_figureTitle('A figure title','propertyName',...
    PropertyValue)
```

See also:

[text](#), [nb_footer](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- alignment
- children
- deleteOption
- extent
- fontName
- fontSize
- fontUnits
- fontWeight
- interpreter
- parent
- placement
- position
- string
- visible
- wrap

- [alignment](#) ↑

The figure title text alignment, default is 'left'. You also have 'center' and 'right'.

- **children** ↑

The children of the object. As MATLAB text object(s).

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **extent** ↑

The extent of the figure title object. Specifies a rectangle that locates in the units of the first child of the parent object. The vector is of the form: [left bottom width height], where left and bottom define the distance from the lower-left corner of the container to the lower-left corner of the rectangle. width and height are the dimensions of the rectangle. Allways in normalized units.

- **fontName** ↑

Font name used for the figure title. 'arial' default.

- **fontSize** ↑

Size of the text. Must be a scalar.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' |
'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The footer font weight, either 'bold', 'normal' or 'light'.

- **interpreter** ↑

Latex interpretation is default, set it to 'none' otherwise.

- **parent** ↑

The parent of the this plotting object. Must be a nb_figure object.

- **placement** ↑

Where to place the figure title in the x-axis direction, either 'center', 'right', 'leftaxes' or the default value 'left'.

- **position** ↑

A 1x2 double with the position of the figure title.

- **string** ↑

The figure title text, as a string or a char with the text. If it is a char, each row will be added as a separate line.

- **visible** ↑

Sets the visibility of the radar plot. {'on'} | 'off'

- **wrap** ↑

Wrap text of footer, i.e. make automatic line breaks.

Methods:

- [get](#)
- [set](#)

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_figureTitle

Input:

- obj : An object of class nb_figureTitle
- propertyName : A string with the propertyName

Output:

- `propertyValue` : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► set ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class `nb_figureTitle`. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- `obj` : An object of class `nb_figureTitle`
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class `nb_figureTitle` with the given properties reset

Examples:

```
obj.set('propertyName',propertyValue,...);
```

Written by Kenneth SÃ¶terhagen Paulsen

■ nb_footer

Go to: [Properties](#) | [Methods](#)

```
obj = nb_footer(string,varargin)
```

Superclasses:

handle

Description:

This is a class for making footers on an nb_figure object. Placed below all elements of the figure

Constructor:

```
obj = nb_footer(string,varargin)

% Input:
- string : The text of the footer, as a string or a char.
    If it is a char, each row will be added as
    a separate lines.

- varargin : ..., 'propertyName', 'PropertyValue', ...
```

Output

```
- obj : An object of class nb_footer
```

Examples:

```
nb_footer('A footer')
f = nb_footer('A footer')
f = nb_footer(char('A footer','New line'))
f = nb_footer('A footer','propertyName',...
    PropertyValue)
```

Written by Kenneth Sætherhagen Paulsen

Properties:

- alignment
- children
- deleteOption
- extent
- fontName
- fontSize
- fontUnits
- fontWeight
- interpreter
- parent
- placement
- position
- string
- visible
- wrap

- alignment ↑

The footer alignment, default is 'left'. You also have 'center' and 'right'.

- children ↑

The children of the object. As MATLAB text object(s).

- deleteOption ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- extent ↑

The extent of the figure title object. Specifies a rectangle that locates in the units of the first child of the parent object. The vector is of the form: [left bottom width height], where left and bottom define the distance from the lower-left corner of the container to the lower-left corner of the rectangle. width and height are the dimensions of the rectangle. Allways in normalized units.

- **fontName** [↑](#)

Font name used for the footer. 'arial' is default.

- **fontSize** [↑](#)

Size of the text. Must be a scalar.

- **fontUnits** [↑](#)

{'points'} | 'normalized' | 'inches' |
'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** [↑](#)

The footer font weight, either 'bold', 'normal' or 'light'.

- **interpreter** [↑](#)

Latex interpretation is default, set it to 'none' otherwise.

- **parent** [↑](#)

The parent of the plotting object. Must be a nb_figure object.

- **placement** [↑](#)

Where to place the footer in the x-axis direction, either 'center', 'right', 'leftaxes' or the default value 'left'.

- **position** [↑](#)

A 1x2 double with the position of the figure title.

- **string** ↑

The footer text, as a string or a char with the text. If it is a char, each row will be added as a separate lines.

- **visible** ↑

Sets the visibility of the radar plot. {'on'} | 'off'.

- **wrap** ↑

Wrap text of footer, i.e. make automatic line breaks.

Methods:

- **get**
 - **set**
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_footer

Input:

- **obj** : An object of class nb_footer
- **propertyName** : A string with the propertyName

Output:

- **propertyValue** : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_footer. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_footer
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_footer with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_gradedFanChart

Go to: [Properties](#) | [Methods](#)

```
obj = nb_gradedFanChart(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for plotting a graded fan chart.

Caution: Use the nb_colormap or the nb_axes.colorMap property to set colors of the fan. If the colormap specifies too few colors they are used to construct an interpolated colormap.

Constructor:

```
nb_gradedFanChart(yData)
nb_gradedFanChart(xData,yData)
nb_gradedFanChart(xData,yData,'propertyName',PropertyValue,...)
handle = nb_gradedFanChart(xData,yData,'propertyName',...
                           PropertyValue,...)
```

Input:

```

- xData      : The xData of the plotted data. Must be of size;
               size(yData,1) x 1 or 1 x size(yData,1)

- yData      : The yData of the plot. The columns are counted
               as different simulations. And on the basis of this
               data the confidence intervals are calculated.

- varargin : ...,'propertyName',PropertyValue,...
```

Output

```
- obj       : An object of class nb_gradedFanChart
```

Examples:

```

obj = nb_gradedFanChart(rand(2,100)); % Provide only y-axis data
obj = nb_gradedFanChart([1,2],rand(2,100));
obj = nb_gradedFanChart(rand(2,100),'propertyName',PropertyValue,...)
obj = nb_gradedFanChart([1,2],rand(2,100),'propertyName',...
                           PropertyValue,...)
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- alpha
- central
- children
- deleteOption
- legendInfo
- lineWidth
- method
- parent
- side
- style
- visible
- xData
- yData

- **alpha** ↑

Transparency of fan chart. A double between 0 and 1. 0 fully transparent, 1 opaque.

- **central** ↑

An nb_line object with the mean, if you set this as a string the mean will not be plotted

- **children** ↑

The children of the plot, as MATLAB patch handles.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **legendInfo** ↑

{'off'} : The mean line will not be included in the legend. 'on' : included it in the legend

- **lineWidth** ↑

The line width of center of the fan. See also property center.

- **method** ↑

If the the data is all in level use 'level', if one is in level and the rest is in diff to the first column use 'diff'. When set to 'level' the data is sorted and transformed to the format needed when set to 'diff'.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **style** ↑

Either 'patch' and 'line'.

- **visible** ↑

Set the visibility of the fan chart.

- **xData** ↑

The x-axis data. Should be a nObs x 1 vector.

- **yData** ↑

Each column of the yData property is counted as one simulation. As a nObs x nSim.

Caution : If this property is set to nObs x 2 double, it will take the data as the upper and lower limit of the graded fan, and the size of the color map (see nb_axes.colorMap) will decide the number points used for the graded fan chart.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► [findClosestNiceNumber](#) ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when `findAxisMethod == 3` is used.

Input:

- `number` : The given number to find a "nice" number close to.
- `scaleNumber` : The scale factor used. I.e. the precision of the algorithm.
- `up` : 1 if rounding up, 0 if rounding down.

Output:

- `niceNumber` : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► [findLimitsAlgo](#) ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_fanChart

Input:

- obj : An object of class nb_fanChart
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('central');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'
```

```
> 'grey','grÃ¥','gr'
> 'purple','lilla','p'
> 'yellow','gul','y'
> 'magenta','m'
> 'white','hvit','w'
> 'cyan','c'
> 'burgundy','burgunder'
> 'sky blue','himmelblÃ¥','hb'
> 'deep blue','mÃ¸rk blÃ¥','db'
> 'water blue','vannblÃ¥','vb'
> 'cool grey','kald grÃ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_fanChart. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_fanChart
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_fanChart with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_graphPanel

Go to: [Properties](#) | [Methods](#)

```
obj = nb_graphPanel(varargin)
```

Superclasses:

handle

Description:

This is class for creating a graph panel, which will have a resize behavioral to keep the font size fixed proportion to the axes (as nb_axes object) of the panel.

Caution : The panel will be centralized in the figure!

Constructor:

```
obj = nb_graphPanel(aspectRatio varargin)
```

Input:

- aspectRatio : Sets the aspect ratio of the graph panel.
Must be a scalar. Give [], if you want to use the default.
- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_graphPanel

Examples:

```
nb_graphPanel  
obj = nb_graphPanel(aspectRatio)  
obj = nb_graphPanel(aspectRatio,...  
'propertyName', PropertyValue,...)
```

See also:

[nb_figure](#), [nb_axes](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- advanced
- aspectRatio
- children
- deleteOption
- figureHandle
- figureTitle
- footer
- panelHandle
- userData

- **advanced** ↑

Indicate if advanced figure is made.

- **aspectRatio** [↑](#)

Sets the aspect ratio of the graph panel. Default is '[4,3]'

- **children** [↑](#)

The children of the figure. Must be of class nb_axes.

Inherited from superclass NB FIGURE

- **deleteOption** [↑](#)

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

Inherited from superclass NB FIGURE

- **figureHandle** [↑](#)

A MATLAB figure handle.

Inherited from superclass NB FIGURE

- **figureTitle** [↑](#)

The title of the figure. Placed above all the figure axes as an nb_figureTitle object.

Inherited from superclass NB FIGURE

- **footer** [↑](#)

The footer placed below all the figure axes as an nb_footer object.

Inherited from superclass NB FIGURE

- **panelHandle** [↑](#)

Handle to the uipanel object of the nb_graphPanel object.

- **userData** [↑](#)

User data

Inherited from superclass NB FIGURE

Methods:

- [get](#)
- [getInnerExtent](#)
- [set](#)

► **get** [↑](#)

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_figure

Input:

- obj : An object of class nb_figure
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getInnerExtent** ↑

```
extent = getInnerExtent(obj,units)
```

Description:

Get inner extent of all children in a nb_figure object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB FIGURE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_figure. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_figure
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_figure with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_hbar

Go to: [Properties](#) | [Methods](#)

```
obj = nb_hbar(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making bar plots with or without shaded bars.

```
nb_hbar(yData)
nb_hbar(xData,yData)
nb_hbar(xData,yData,'PropertyName',PropertyValue,...)
handle = nb_hbar(xData,yData,'PropertyName',PropertyValue,...)
```

The handle have set and get methods as all MATLAB graph handles.

Caution: All the children of this object is of class nb_patch

Constructor:

```
obj = nb_hbar(xData,yData,varargin)
```

Input:

- xData : The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)
- yData : The yData of the plot. The columns are counted as seperate variables
- varargin : ..., 'PropertyName', PropertyValue, ...

Output

- obj : An object of class nb_hbar

Examples:

```
obj = nb_hbar([1,2]); % Provide only y-axis data  
obj = nb_hbar([1,2],[1,2]);  
obj = nb_hbar([1,2],'propertyName',PropertyValue,...)  
obj = nb_hbar([1,2],[1,2],'propertyName',PropertyValue,...)
```

See also:

[hbar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- alpha1
- alpha2
- barWidth
- baseValue
- baseline
- blend
- cData
- children
- deleteOption
- direction
- edgeColor
- legendInfo
- line
- lineStyle
- lineWidth
- parent
- scale
- shadeColor
- shaded
- side
- style
- sumTo
- visible
- xData
- yData

- **alpha1** ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- **alpha2** ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- **barWidth** ↑

Width of the bars. Must be a scalar.

- **baseValue** ↑

The base value of the plot. Must be a scalar.

- **baseline** ↑

A nb_horizontalLine handle (object). Use the set and get methods of this handle to change the baseline properties

- **blend** ↑

When shaded is used, and this option is set to true, it will do alpha blending with shadeColor instead of shading.

- **cData** ↑

Color of the plotted data. A matrix;
size(yData,2) x 3 (RGB color). Or a cellstr
with the color names. (With size 1 x size(yData,2))

- **children** ↑

All the handles of the bar plot.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be
delete and all that is plotted by this object are removed
from the figure it is plotted on. If on the other hand
'only' is given, it will just delete the object.

- **direction** ↑

The direction of the shading; {'north'} | 'south' |
'east' | 'west'.

- **edgeColor** ↑

The color of the edge of the bars. In RGB colors
or a string with the color names. Can also be
'none' ; no edgeLine or 'same' ; same color as
the base color for each bar. Can also be a
cellstr with either the RGB colors or color
names with size 1 x size(yData,2).

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in
the legend.

- **line** ↑

A nb_line handle representing the sum of the stacked bars
when 'style' is set to 'dec'.

- **lineStyle** ↑

Sets the line style(s) of the edge of the bars.
Either a string or a cellstr (which must have
the size 1 x size(yData,2)).

- **lineWidth** ↑

The line width of the edge of the bars. Must be a scalar.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **scale** ↑

Sets the tightness of the bars plotted. (Stacked)

- **shadeColor** ↑

The color (RGB) the shaded bar plots are interpolated with. Default is white

- **shaded** ↑

Index of which data should be shaded. A matrix with size; size(yData,1) x 1 or 1 x size(yData,1) or size(yData,1) x size(yData,2)

- **side** ↑

From which sides the horizontal bar should start. 'left' or 'right'

- **style** ↑

'stacked' | {'grouped'} | 'dec'. When 'dec' is given the bars will be stacked but there will also be plotted a line with the sum. (It is not possible to use the sumTo option at the same time)

- **sumTo** ↑

If given the sum of the bars for each period will sum up to this number. (only when style is set to 'stacked')
E.g. if set 100, each bar for each variable will be given as percentage share of the total sum.

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **xData** ↑

The xData of the plotted data. Must be of size; size(yData,1) x 1 or 1 x size(yData,1)

- **yData** ↑

The yData of the plot. The columns are counted as seperate variables.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► [findClosestNiceNumber](#) ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when `findAxisMethod == 3` is used.

Input:

- `number` : The given number to find a "nice" number close to.
- `scaleNumber` : The scale factor used. I.e. the precision of the algorithm.
- `up` : 1 if rounding up, 0 if rounding down.

Output:

- `niceNumber` : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► [findLimitsAlgo](#) ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_bar

Input:

- obj : An object of class nb_bar
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nr','g'  
> 'orange','o'  
> 'light blue','lys blÅ','lb'  
> 'black','svart','k'
```

```
> 'grey','grÃ¥','gr'
> 'purple','lilla','p'
> 'yellow','gul','y'
> 'magenta','m'
> 'white','hvit','w'
> 'cyan','c'
> 'burgundy','burgunder'
> 'sky blue','himmelblÃ¥','hb'
> 'deep blue','mÃ¸rk blÃ¥','db'
> 'water blue','vannblÃ¥','vb'
> 'cool grey','kald grÃ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_bar. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_bar
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_bar with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_highlight

Go to: [Properties](#) | [Methods](#)

```
obj = nb_highlight(xData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

A class for making highlighted colored areas in the background of a plot

The parent of this axes must be a nb_axes handle.

This colored background will automatically adjust to changes in the y-axis limits.

Constructor:

```
obj = nb_highlight(xData,varargin)
```

Input:

- xData : A 1 x 2 double with the x-axis limits of the highlighted area.

Optional input:

- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_highlight

Examples:

```
nb_highlight(xData)
```

```
nb_highlight(xData,'propertyName',PropertyValue,...)
```

```
handle = nb_highlight(xData,'propertyName',PropertyValue,...)
```

See also:

[nb_patch](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData • children • deleteOption • legendInfo • parent
- visible • xData

- cData ↑

Must be a 1 x 3 double with the RGB colors or a string with the color name. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- children ↑

The child of this handle. As a nb_patch handle.

- deleteOption ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- legendInfo ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- parent ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- visible ↑

Sets the visibility of the radar plot. {'on'} | 'off'.

- xData ↑

A 1 x 2 double with the x-axis limits of the highlighted area, or a logical vector with size N x 1. In the case xData is given as a logical vector it will highlight the periods where the elements are true.

Methods:

- findClosestNiceNumber • findLimitsAlgo • get • getDefaultColors
- interpretColor • set

► findClosestNiceNumber ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem

```
    when graphing small number)
> 2: Add some space in both direction of the
y-axis. (The limits will seldom be nice
numbers.)
> 3: Uses the submethod findClosestNiceNumber of
the class to decide on the limits. (Also this
method have can have problems with data which
is not symmetric around the zero line.)
> 4 : Uses the MATLAB algorithm for finding the
axis limits.
```

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_highlight

Input:

- obj : An object of class nb_highlight
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'
```

```
> 'water blue','vannblÅ¥','vb'
> 'cool grey','kald grÅ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_highlight. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_highlight
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_highlight with the given properties reset

Examples:

```
obj.set('propertyName',propertyValue,...);
```

Written by Kenneth SÃ¶terhagen Paulsen

■ **nb_horizontalLine**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_horizontalLine(yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making horizontal lines which spans all of the x-axis.

If the parent is a nb_axes object:

- The line will automatically update the length of the line, so it spans all the x-axis.

If the parent is a MATLAB axes handle (object):

- The line will not update itself automatically and you need to use the update() method of the object to make it fit to the axes handle (If it is changed)

Constructor:

```
obj = nb_horizontalLine(yData,varargin)
```

Input:

- yData : The y-axis value to place the horizontal line
- varargin : ..., 'propertyName',PropertyValue, ...

Output

- obj : an object of class nb_horizontalLine

Examples:

```
nb_horizontalLine(yData)
obj = nb_horizontalLine(yData)
obj = nb_horizontalLine(yData,'propertyName',...
                           PropertyValue,...)
```

See also:

[nb_line](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData
- children
- deleteOption
- legendInfo
- lineStyle
- lineWidth
- marker
- markerEdgeColor
- markerFaceColor
- markerSize
- parent
- side
- visible
- yData

- **cData** ↑

The color data of the line plotted. Must be of size; 1 x 3 with the RGB colors or a string with the color name.

- **children** ↑

The children of this object. As an nb_line object.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-' | 'none'

- **lineWidth** ↑

The line width of the plotted data. Must be a scalar.
Default is 1.

- **marker** ↑

The markers of the lines. Either a string or a cell array with size as the number of lines to be plotted. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerSize** ↑

Size of the markers. Must be a integer. Default is 9.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **yData** ↑

A scalar with the y-axis value for where to place the horizontal line.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefauleColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number,...  
                                                scaleNumber,up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- **number** : The given number to find a "nice" number close to.
- **scaleNumber** : The scal factor used. I.e. the precision of the algorithm.
- **up** : 1 if rounding up, 0 if rounding down.

Output:

- **niceNumber** : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_horizontalLine

Input:

- obj : An object of class nb_horizontalLine
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ!terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_horizontalLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_horizontalLine
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_horizontalLine with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_image

Go to: [Properties](#) | [Methods](#)

```
obj = nb_image(cData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for plotting images.

The handle have set and get methods as all MATLAB graph handles.

Constructor:

```
obj = nb_image(cData,'propertyName',PropertyValue,...)
```

Input:

- cData : The data of the image. Must be a double of size M x N (mapped into the colormap) or M x N x 3 (RGB).

In the case that the parent is of class nb_axes the current colormap can be set by nb_colormap, otherwise use colormap.

```
- varargin : ...,'propertyName',PropertyValue,...
```

Output:

```
- obj : An object of class nb_image
```

Examples:

```
nb_image(rand(5,5));  
obj = nb_image(rand(5,5,3)*255);
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- appendWhite
- cData
- children
- deleteOption
- legendInfo
- parent
- side
- visible

- **appendWhite** ↑

When true, white is added to the colormap property before interpreting the cData. Default is true.

- **cData** ↑

The data of the image. Must be a double of size M x N (mapped into the colormap) or M x N x 3 (RGB).

In the case that the parent is of class nb_axes the current colormap can be set by nb_colormap

- **children** ↑

All the handles of the area plot, as nb_patch objects

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes (Which is the default).

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- **number** : The given number to find a "nice" number close to.
- **scaleNumber** : The scale factor used. I.e. the precision of the algorithm.
- **up** : 1 if rounding up, 0 if rounding down.

Output:

- **niceNumber** : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get ↑**

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_area

Input:

- obj : An object of class nb_area
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ;terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

set(obj,varargin)

Description:

Sets the properties of an object of class nb_area. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_area
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_area with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_keyEvent

Go to: [Properties](#) | [Methods](#)

The data that get sent when the nb_figure events keyPressed and keyReleased are triggered.

Superclasses:

event.EventData

See also:

[nb_figure](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- Character • EventName • Key • Modifier • Source

- [Character ↑](#)

nb_keyEvent/Character is a property.

- [EventName ↑](#)

event.EventData.EventName – Name of event

```
Documentation for event.EventData.EventName
doc event.EventData.EventName
```

- [Key ↑](#)

nb_keyEvent/Key is a property.

- **Modifier** ↑

nb_keyEvent/Modifier is a property.

- **Source** ↑

event.EventData.Source - Event source

Documentation for event.EventData.Source
doc event.EventData.Source

Methods:

■ **nb_legend**

Go to: [Properties](#) | [Methods](#)

obj = nb_legend(parent, legends, varargin)

Superclasses:

handle, nb_annotation, nb_movableAnnotation

Description:

This is a class for making legend of plots

I strongly recommend to use this class instead of the MATLAB legend command if you use the nb_* plot functions.

Caution : - When the type property of this class is set to 'matlab', the lineStyle '---', will be given as '-'.

- When the 'type' property of this class is set to 'nb' it is not possible to move the legend by click and drag, as the normal MATLAB legend.
- If you create more legends of a plot the first legends will not longer be moveable. (Even if the type property is set to 'MATLAB')

Constructor:

obj = nb_legend(parent, legends, varargin)

Input:

- parent : A nb_axes handle
- legends : A cellstr of the legends of the plot. Say you do not want to plot the legend of a plotted

variable you can type '' at its location in the cellstr, and it will not be shown in the legend.

- varargin : ...,'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_legend (handle)

Examples:

```
nb_legend(ax,legends)
nb_legend(ax,legend,'propertyName',PropertyValue,...)
leg = nb_legend(ax,legend,'propertyName',PropertyValue,...)
```

See also:

[legend](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|----------------------------------|--------------------------------|-----------------------------|-------------------------------|------------------------------|
| • box | • children | • color | • columnWidth | • columns |
| • copyOption | • deleteOption | • extent | • fakeLegends | • fast |
| • fontColor | • fontName | • fontSize | • fontUnits | • fontWeight |
| • interpreter | • legends | • listeners | • location | • normalized |
| • objectToNotify | • parent | • pointer | • position | • reorder |
| • selected | • space | • visible | | |

- **box** ↑

If you want a box around the legend or not. 'on' | 'off'.
'on' is default. If you set it to 'off' the movability
of the legend is destroyed

- **children** ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **color** ↑

Background color. Either a string with the color name or
a 1x3 double with the RGB colors. Default is 'none'.

- **columnWidth** ↑

Sets the column width of the legend (Only when dealing
with more than one column). If empty they are try to
adjust to the text.

- **columns** ↑

Number of columns of the legend.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **extent** ↑

position rectangle (read-only)

Position and size of the legend. A four-element vector that defines the size and position of the legend:

[left,bottom,width,height]

left and bottom are the x- and y-coordinates of the lower left corner of the text extent. The units are normalized.

- **fakeLegends** ↑

Fake legends added to the legend. Must be given as nb_legendDetails vector with size 1xM.

- **fast** ↑

Indicator if fast legend is tried to be used. If true moving the legend is not possible.

- **fontColor** ↑

Sets the font color(s) of the legend text. Must either be a 1x3 double or a string with the color name of all legend text objects, or a M x 3 double or a cellstr array with size 1 x M with color names to use of each legend text object.

- **fontName** ↑

The font name used for the text of the legend. As a string.

- **fontSize** ↑

Font size of the text in the legend. As a scalar.

- **fontUnits** ↑

```
{'points'} | 'normalized' | 'inches' |  
'centimeters' | 'pixels'
```

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

Font weight of the text in the legend. As a string.

- **interpreter** ↑

```
{'none'} | 'tex' | 'latex' . See the text properties for  
more.
```

- **legends** ↑

The legends of the plot. Given as a cellstr. Say you don't want to plot the legend of a plotted variable you can type '' at its location in the cellstr, and it will not be shown in the legend.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass `NB_ANNOTATION`

- **location** ↑

The location of the legend; '`west`' | {'`northwest`'} | '`north`' | '`northeast`' | '`east`' | '`southeast`' | '`south`' | '`southwest`' | '`center`' | '`below`' | '`middle`' | '`outsideright`' | '`outsiderighttop`'. When adding a legend to a `nb_pie` plot the '`pie`' option could also be used.

- **normalized** ↑

Indicate if the font when `fontUnits` is set to '`normalized`' should be normalized to the axes ('`axes`') or to the figure ('`figure`'), i.e. the default axes position [0.1 0.1 0.8 0.8].

- **objectToNotify** ↑

Set the object to notify. Must contain the property legPosition. Default is []. Deprecated property!

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **position** ↑

Set the position of the legend 1x2 double. I.e;
[xLeftPosition,yLowerPosition]. Only an option when the
type property is set to 'nb'.

- **reorder** ↑

Set this property to reorder the legend. Either
a string with {'default'} | 'inverse' or a
double with how to reorder the legend. E.g. if
you have 3 legends to plot the default index
will be [1,2,3], the inverse ('inv') will be
[3,2,1]. If you want to decide that the 3.
legend should be first, then the 1. and 2., you
can type in [3,1,2].

Caution : If a double is given it must have as
many elements as there is plotted
legends. I.e. if you give an empty
string to one of the legends it will
be excluded. Say you provide the
property legends as {'s','','t','f'}
then the double must have size 1x3.

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **space** ↑

Vertical space between the text in the legend. Must be
scalar. Default is

- **visible** ↑

Sets the visibility of the legend. {'on'} | 'off'.

Methods:

- [get](#)
 - [set](#)
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_legend

Input:

- obj : An object of class nb_legend
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_legend. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_legend
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_legend with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_legendDetails

Go to: [Properties](#) | [Methods](#)

```
obj = nb_legendDetails()
```

Description:

This is a class for storing information of how the legend should look.

Constructor:

```
obj = nb_legendDetails()
```

Input:

No inputs.

Output:

- obj : An object of class nb_legendDetails

Examples:

See also:

[nb_legend](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | |
|-----------------------|-----------------------|----------------------|
| • fontColor | • lineColor | • lineMarker |
| • lineMarkerEdgeColor | • lineMarkerFaceColor | • lineMarkerSize |
| • lineStyle | • lineWidth | • patchColor |
| • patchDirection | • patchEdgeColor | • patchEdgeLineStyle |
| • patchEdgeLineWidth | • patchFaceAlpha | • patchFaceLighting |
| • side | • type | |

- [fontColor](#) ↑

nb_legendDetails/fontColor is a property.

- **lineColor** ↑

nb_legendDetails/lineColor is a property.

- **lineMarker** ↑

nb_legendDetails/lineMarker is a property.

- **lineMarkerEdgeColor** ↑

nb_legendDetails/lineMarkerEdgeColor is a property.

- **lineMarkerFaceColor** ↑

nb_legendDetails/lineMarkerFaceColor is a property.

- **lineMarkerSize** ↑

nb_legendDetails/lineMarkerSize is a property.

- **lineStyle** ↑

nb_legendDetails/lineStyle is a property.

- **lineWidth** ↑

nb_legendDetails/lineWidth is a property.

- **patchColor** ↑

nb_legendDetails/patchColor is a property.

- **patchDirection** ↑

nb_legendDetails/patchDirection is a property.

- **patchEdgeColor** ↑

nb_legendDetails/patchEdgeColor is a property.

- **patchEdgeLineStyle** ↑

nb_legendDetails/patchEdgeLineStyle is a property.

- **patchEdgeLineWidth** ↑

nb_legendDetails/patchEdgeLineWidth is a property.

- **patchFaceAlpha** ↑

nb_legendDetails/patchFaceAlpha is a property.

- **patchFaceLighting** ↑

nb_legendDetails/patchFaceLighting is a property.

- **side** ↑

nb_legendDetails side is a property.

- **type** ↑

nb_legendDetails/type is a property.

Methods:

■ **nb_line**

Go to: [Properties](#) | [Methods](#)

obj = nb_line(xData,yData,varargin)

Superclasses:

handle, nb_plotHandle

Description:

This is a class for plotting a line. (only 2D)

Extra functionalities:

- The line style '---' is supported.

Caution: Some functionalities of the MATLAB line command is not supported.

Important:

- z-dimension is not supported
- Only possible to plot one line at a time.

Constructor:

obj = nb_line(xData,yData,varargin)

Input:

- xData : The x-axis values of the plotted line. If only one input is given to this constructor it will be interpreted as the yData input. (Then the xData will be 1:size(yData,1)). Must be a double vector.

- yData : The y-axis values of the plotted line. Must be a double vector.

Optional input:

- varargin : ..., 'propertyName', PropertyValue, ...

Output:

- obj : An object of class nb_line

Examples:

```
nb_line([0,2]);
```

same as

```
nb_line([0,1],[0,2]);
```

```
nb_line([0,1],[0,2], 'propertyName', PropertyValue, ...);
```

e.g.

```
nb_line([0,1],[0,2], 'cData', {'red'}, 'lineWidth', 1.5);
```

Return the handle to the plotted line:

```
l = nb_line([0,1],[0,2], 'cData', {'red'}, 'lineWidth', 1.5);
```

Which you can use to set and get properties:

```
l.set('lineWidth', 1);
lw = l.get('lineWidth');
```

See also:

[line](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [cData](#)
- [children](#)
- [clipping](#)
- [dashLength](#)
- [deleteOption](#)
- [gapLength](#)
- [legendInfo](#)
- [lineStyle](#)
- [lineWidth](#)
- [marker](#)
- [markerEdgeColor](#)
- [markerFaceColor](#)
- [markerSize](#)
- [parent](#)
- [side](#)
- [visible](#)
- [xData](#)
- [xLim](#)
- [yData](#)
- [yLim](#)

- **cData** ↑

The color data of the line plotted. Must be of size;
1 x 3. With the RGB colors. Or a string with the name
of the color. See the method `interpretColor` of the
nb_plotHandle class for more on the supported color
names.

- **children** ↑

The children of the handle. As line handle(s).

- **clipping** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

- **dashLength** ↑

Lenght of the dashes of the '---' lineStyle.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **gapLength** ↑

Lenght of the gaps of the '---' lineStyle

- **legendInfo** ↑

'off' : Not included in the legend. {'on'} : included in the legend

- **lineStyle** ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-.' | 'none'

- **lineWidth** ↑

The line width of the plotted data. Must be a scalar. Default is 2.5.

- **marker** ↑

The markers of the lines. Either a string or a cell array with size as the number of lines to be plotted. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerSize** ↑

Size of the markers. Default is 9.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **xData** ↑

The x-axis data. As a double vector.

- **xLim** ↑

The x-axis limits as 1x2 double. Default is to use the axes limits (If they are smaller then the data limits, the data limits will be used instead). This property is only used for the line style '---'

- **yData** ↑

The y-axis data. As a double vector.

- **yLim** ↑

The y-axis limits as 1x2 double. Default is to use the axes limits (If they are smaller then the data limits, the data limits will be used instead). This property is only used for the line style '---'

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► [findClosestNiceNumber](#) ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► [findLimitsAlgo](#) ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_line

Input:

- obj : An object of class nb_line
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nr','g'  
> 'orange','o'  
> 'light blue','lys blÅ','lb'  
> 'black','svart','k'
```

```
> 'grey','grÃ¥','gr'
> 'purple','lilla','p'
> 'yellow','gul','y'
> 'magenta','m'
> 'white','hvit','w'
> 'cyan','c'
> 'burgundy','burgunder'
> 'sky blue','himmelblÃ¥','hb'
> 'deep blue','mÃ¸rk blÃ¥','db'
> 'water blue','vannblÃ¥','vb'
> 'cool grey','kald grÃ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_line. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_line
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_line with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_mouseOverObjectEvent

Go to: [Properties](#) | [Methods](#)

The data that get sent when a mouseOverObject event is triggered by an object of class nb_notifiesMouseOverObject.

Superclasses:

```
event.EventData
```

See also:

[nb_notifiesMouseOverObject](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [EventName](#)
- [Source](#)
- [currentPoint](#)
- [h](#)
- [value](#)
- [x](#)
- [y](#)

- [EventName ↑](#)

```
event.EventData.EventName - Name of event
```

Documentation for `event.EventData.EventName`
doc `event.EventData.EventName`

- [Source ↑](#)

```
event.EventData.Source - Event source
```

Documentation for `event.EventData.Source`
doc `event.EventData.Source`

- [currentPoint ↑](#)

Current point in axes data units. As a 1x2 double.

- [h ↑](#)

The children index, as an integer

- **value ↑**

The value of the data point the mouse is over. As a scalar double

- **x ↑**

The x-data index, as an integer.

- **y ↑**

The y-data index, as an integer.

Methods:

■ **nb_patch**

Go to: [Properties](#) | [Methods](#)

obj = nb_patch(xData,yData,cData,varargin)

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making patches.

This class makes shaded patches which doesn't return a warning when printed by the export_fig function developed by Oliver Woodford

Caution :

- Not all functionalities of the patch class is supported by this class.
- The transparency options is not supported by the export_fig function when the renderer is set to 'painters' (Which is default for .eps and .pdf files). Use 'openGL'.
- The lighting options is only supported by the renderer 'openGL'
- This class does not support zData.
- Only rectangular patches will be shaded correctly at the moment

Constructor:

```
obj = nb_patch(xData,yData,cData,varargin)
```

Input:

- xData : The x-axis values of the plotted patch. If only one input is given to this constructor it will be interpreted as the yData input. (Then the xData will be 1:size(yData,1)). Must be a double vector.
- yData : The yData of the plot. Must be a double vector.
- cData : The RGB color of the patch, if it has two values the color will be interpolated. The first RGB values will at the bottom of the patch.
Must be a 1 x 3 double (or 2 x 3 double vector when using shading) with the RGB colors or a cellstr with the color names. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

If not provided the color 'black' is used.

Optional input:

- varargin : ..., 'propertyName', PropertyValue, ...

Output

- obj : An object of class nb_patch

Examples:

```
nb_patch(yData)
nb_patch(xData,yData)
nb_patch(xData,yData,cData)
nb_patch(xData,yData,cData,'propertyName',PropertyValue,...)
handle = nb_patch(xData,yData,cData,'propertyName',...
                  PropertyValue,...)
```

See also:

[patch](#), [surface](#), [fill](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData
 - children
 - clipping
 - deleteOption
 - direction
 - edgeColor
 - faceAlpha
 - faceLighting
 - legendInfo
 - lineWidth
 - parent
 - side
 - visible
 - xData
 - yData
- **cData** ↑

The RGB color of the patch, if it has two values the color will be interpolated. The first RGB values will at the bottom of the patch.

Must be a 1 x 3 double (or 2 x 3 double vector when using shading) with the RGB colors or a string with the color name. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- **children** [↑](#)

All the children (patch and line handles) of the `nb_patch` object

- **clipping** [↑](#)

'on' or 'off'. Clipping to axes rectangle. When Clipping is on, MATLAB does not display any portion of the patch outside the axes rectangle.

- **deleteOption** [↑](#)

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

- **direction** [↑](#)

Direction of the shading; {'north'} | 'south' | 'west' | 'east'

- **edgeColor** [↑](#)

The color of the edge of the bars. In RGB colors | 'none' : No edgeLine | 'same' : Same color as the base color for each bar | a string with the color name.

- **faceAlpha** [↑](#)

Transparency of the patch face. Only the scalar option is supported. A single non-NaN value between 0 and 1 that controls the transparency of all the faces of the object. 1 (the default) means fully opaque and 0 means completely transparent (invisible)

- **faceLighting** [↑](#)

See the patch properties for more on this option

- **legendInfo** [↑](#)

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

The line style of the edge of the patch.

- **lineWidth** ↑

The line width of the edge of the patch.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **visible** ↑

Patch object visibility. 'on' : The patch is visible.
'off' : The patch is not visible, but still exists, and you can query and set its properties.

- **xData** ↑

The x values of the patch. Same as for the MATLAB patch class

- **yData** ↑

The y values of the patch. Same as for the MATLAB patch class

Methods :

- [findClosestNiceNumber](#)
- [findLimitsAlgo](#)
- [get](#)
- [getDefaultColors](#)
- [interpretColor](#)
- [set](#)

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of

the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
> 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

propertyValue = get(obj,propertyName)

Description:

Get a property of an object of class nb_patch

Input:

- obj : An object of class nb_patch
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

propertyValue = obj.get('children');

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

defaultColors = nb_plotHandle.getDefaultColors(dataSize)

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

colors = nb_plotHandle.interpretColor(colors)

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÃ,d','r'  
> 'green','grÃ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÃ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÃ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÃ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colormames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

set(obj,varargin)

Description:

Sets the properties of an object of class nb_patch. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_patch
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_patch with the given properties reset

Examples:

obj.set('propertyName',PropertyValue,...);

Written by Kenneth Sæterhagen Paulsen

■ **nb_pie**

Go to: [Properties](#) | [Methods](#)

obj = nb_pie(yData,varargin)

Superclasses:

```
handle, nb_plotHandle
```

Description:

This is a class for making pie charts in 2D.

This class will make the pie chart automatically fit the axes of the plot.

I recommend to use the nb_* classes for the legend, titles and so on.

Constructor:

```
obj = nb_pie(yData,varargin)
```

Input:

- yData : The data to plot. min(size(yData)) must be 1.

Optional input:

- varargin : 'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_pie (handle).

Examples:

```
nb_pie(yData)
nb_pie(yData,'propertyName',PropertyValue,...)
handle = nb_pie(yData,'propertyName',PropertyValue,...)
```

See also:

[pie](#), [pie3](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- axisVisible
- bite
- cData
- children
- deleteOption
- edgeColor
- explode
- fontColor
- fontName
- fontSize
- fontUnits
- fontWeight
- labelHandles
- labels
- labelsExtension
- legendInfo
- lineStyle
- lineWidth
- location
- noLabels
- origoPosition
- parent
- visible
- yData

- axisVisible ↑

Make the axes box visible ('on') or not ('off'). 'on' is default.

- **bite** ↑

A logical array with ones for the pies to bite. I.e. [0,0,1,0]; Must be of size 1 x nSlices.

- **cData** ↑

The colors of the plotted data. Must be of size nSlices x 3. (Double with the RGB colors), or a cellstr with the color names. Must be of size 1 x length(yData). See the method nb_plotHandle.interpretColor for more on the supported color names.

- **children** ↑

The handles of all the children of the plot.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **edgeColor** ↑

The color of the edge of the pies. Must be of size; 1 x 3. (Double with the RGB colors), or a one line char with the color name. See the method nb_plotHandle.interpretColor for more on the supported color names.

- **explode** ↑

A logical array with ones for the pies to explode. I.e. [0,0,1,0]; Must be of size 1 x nSlices.

- **fontColor** ↑

Color of labels. Must be double with size 1 x 3 (RGB colors), or a string with the name of the color. See the method nb_plotHandle.interpretColor for more on the supported color names.

- **fontName** ↑

Name of the font used. Default is 'arial'.

- **fontSize** ↑

Font size of the text on the plot. Default is 12.

- **fontUnits** ↑

```
{'points'} | 'normalized' | 'inches' |  
'centimeters' | 'pixels'
```

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

`normalized` - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

`pixels`, `inches`, `centimeters`, and `points`: Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

Font weight of the text on the plot. Default is 'normal'.

- **labelHandles** ↑

Handles to the labels of the pie plot. As MATLAB text objects.

- **labels** ↑

The labels of the pies, must be a `cellstr` with `size`; `length(yData)`. If not given the labels will be the shares of each pie in percent.

- **labelsExtension** ↑

If you not give the labels you can set the extension of the labels. Default is '%'. The extension will be added to the value of the data. If '%' is given the data will be in relative shares in percent.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend.

- **lineStyle** ↑

Line style of the edge.

- **lineWidth** ↑

Line width of the edge.

- **location** ↑

{'west'} | 'east' | 'center'. The location of the pie chart in the current axes.

- **noLabels** ↑

Force there to be no labels of the pies.

- **origoPosition** ↑

Set the origo position. Default is to use the location property. Where 'west' is [-0.5,0], 'center' is [0,0] and 'east' is [0.5,0]. Must either be empty or a 1x2 double.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **yData** ↑

The data to plot. Must be a double vector of size 1 x nSlices or nSlices x 1.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefalutColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...  
scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- **number** : The given number to find a "nice" number close to.
- **scaleNumber** : The scale factor used. I.e. the precision of the algorithm.
- **up** : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► findLimitsAlgo ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_pie

Input:

- obj : An object of class nb_pie
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_pie. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_pie
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_pie with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ **nb_plot**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_plot(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is A class for plotting data by lines (2D). Much the same as the MATLAB plot function, but with some extensions (and some limitation)

Extended functionalitites:

- New line style added '---'. Longer dashed of the dashed line
- Choose the linestyle of all the plotted lines. By a cell.
- Choose the linecolor of all the plotted lines directly to

the class, and not through the default options. By a mx3 double with the RGB colors of the m lines

- Makes it possible to set a nb_axes handle as parent (default). Which makes the plot nicer. It is also possible to add shaded background. See nb_axes for more.

Constructor:

```
nb_plot(yData)
nb_plot(xData,yData)
nb_plot(xData,yData,'propertyName',propertyValue,...)
handle = nb_plot(xData,yData,'propertyName',propertyValue,...)
```

Input:

- xData : The x-axis values of the plotted line. If only one input is given to this constructor it will be interpreted as the yData input. (Then the xData will be 1:size(yData,1))
- yData : The y-axis values of the plotted line. A double matrix. (1. dimension the number of observations and 2. dimension the number of variables.)

Optional input:

- varargin : 'propertyName',propertyValue,...

Output:

- obj : An object of class nb_plot

Examples:

```
nb_plot([0,2]);
```

same as

```
nb_plot([0,1],[0,2;2,3]);
```

```
nb_plot([0,1],[0,2],'propertyName',propertyValue,...);
```

e.g.

```
nb_plot([0,1],[0,2;2,3],'cData',{'red','green'},...
'lineWidth',1.5);
```

Return the handle to the plotted line:

```
l = nb_plot([0,1],[0,2;2,3],'cData',{'red','green'},...
'lineWidth',1.5);
```

Which you can use to set and get properties:

```
l.set('cData',{'black','red'});
lw = l.get('lineWidth');
```

See also:

plot

Written by Kenneth Sæterhagen Paulsen

Properties:

- `cData`
- `children`
- `clipping`
- `deleteOption`
- `legendInfo`
- `lineStyle`
- `lineWidth`
- `marker`
- `markerEdgeColor`
- `markerFaceColor`
- `markerSize`
- `parent`
- `side`
- `visible`
- `xData`
- `yData`

- `cData` ↑

The color data of the line plotted. Must be of size; `size(yData,2) x 3`. With the RGB colors. or a cellstr with the color names. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- `children` ↑

The children of the handle. As `nb_line` handles

- `clipping` ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

- `deleteOption` ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- `legendInfo` ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- `lineStyle` ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-' | 'none'

- `lineWidth` ↑

The line width of the plotted data. Must be a scalar.
Default is 2.5.

- **marker** ↑

The markers of the lines. Either a string or
a cell array with size as the number of lines
to be plotted. See marker property of the
MATLAB line function for more on the options
of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' |
{'auto'} | a string with the color name.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' |
{'auto'} | a string with the color name.

- **markerSize** ↑

Size of the markers. Default is 9.

- **parent** ↑

The parent axes you wan to plot on, if not given it will
be plotted in a new nb_axes handle. Either a axes or a
nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the
parent is off class nb_axes. (Which is the default)

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'

- **xData** ↑

The x-axis data. As a double.

- **yData** ↑

The y-axis data. As a double.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
-

► [findClosestNiceNumber](#) ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- `number` : The given number to find a "nice" number close to.
- `scaleNumber` : The scale factor used. I.e. the precision of the algorithm.
- `up` : 1 if rounding up, 0 if rounding down.

Output:

- `niceNumber` : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► [findLimitsAlgo](#) ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth SÅterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► get ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_plot

Input:

- obj : An object of class nb_plot
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'
```

```
> 'grey','grÃ¥','gr'
> 'purple','lilla','p'
> 'yellow','gul','y'
> 'magenta','m'
> 'white','hvit','w'
> 'cyan','c'
> 'burgundy','burgunder'
> 'sky blue','himmelblÃ¥','hb'
> 'deep blue','mÃ¸rk blÃ¥','db'
> 'water blue','vannblÃ¥','vb'
> 'cool grey','kald grÃ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_plot. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_plot
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_plot with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_plotBarAtEnd

Go to: [Properties](#) | [Methods](#)

```
obj = nb_plotBarAtEnd(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle, nb_plot

Description:

This is A class for plotting data by lines (2D), but then switch to bar plot at the end.

Constructor:

```
nb_plotBarAtEnd(yData)
nb_plotBarAtEnd(xData,yData)
nb_plotBarAtEnd(xData,yData,'PropertyName',PropertyValue,...)
h = nb_plotBarAtEnd(xData,yData,'PropertyName',PropertyValue,...)
```

Input:

- xData : The x-axis values of the plotted line. If only one input is given to this constructor it will be interpreted as the yData input. (Then the xData will be 1:size(yData,1))
- yData : The y-axis values of the plotted line. A double matrix. (1. dimension the number of observations and 2. dimension the number of variables.)

Optional input:

- varargin : 'PropertyName', PropertyValue, ...

Output:

- obj : An object of class nb_plotBarAtEnd

Examples:

```
h = nb_plotBarAtEnd(1:150,rand(150,1),'lineStop',40,'barPeriods',[50,100,150]);
```

See also:

[plot](#), [nb_plot](#), [nb_bar](#), [bar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [barPeriods](#)
- [clipping](#)
- [lineStop](#)
- [markerEdgeColor](#)
- [side](#)
- [cData](#)
- [deleteOption](#)
- [lineStyle](#)
- [markerFaceColor](#)
- [visible](#)
- [cDataBar](#)
- [endBarWidth](#)
- [lineWidth](#)
- [markerSize](#)
- [xData](#)
- [children](#)
- [legendInfo](#)
- [marker](#)
- [parent](#)
- [yData](#)

- **barPeriods** ↑

The periods for where to plot the bar. E.g 3 or [3,4]

- **cData** ↑

The color data of the line plotted. Must be of size; size(yData,2) x 3. With the RGB colors. or a cellstr with the color names. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

Inherited from superclass NB_PLOT

- **cDataBar** ↑

A string with the color specification to use for the bars
({'nb'} | 'red' | 'green' | 'yellow' | '')

- **children** ↑

The children of the handle. As nb_line handles

Inherited from superclass NB_PLOT

- **clipping** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

Inherited from superclass NB_PLOT

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

Inherited from superclass NB_PLOT

- **endBarWidth** ↑

The width of the bars at the end.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

Inherited from superclass NB_PLOT

- **lineStop** ↑

The period that the line should be stopped. E.g. 2.

- **lineStyle** ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-' | 'none'

Inherited from superclass NB_PLOT

- **lineWidth** ↑

The line width of the plotted data. Must be a scalar. Default is 2.5.

Inherited from superclass NB_PLOT

- **marker** ↑

The markers of the lines. Either a string or a cell array with size as the number of lines to be plotted. See marker property of the MATLAB line function for more on the options of this property

Inherited from superclass NB_PLOT

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

Inherited from superclass NB_PLOT

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

Inherited from superclass NB_PLOT

- **markerSize** ↑

Size of the markers. Default is 9.

Inherited from superclass NB_PLOT

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

Inherited from superclass NB_PLOT

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'

Inherited from superclass NB_PLOT

- **xData** ↑

The x-axis data. As a double.

Inherited from superclass NB_PLOT

- **yData** ↑

The y-axis data. As a double.

Inherited from superclass NB_PLOT

Methods:

- [findClosestNiceNumber](#)
- [findLimitsAlgo](#)
- [get](#)
- [getDefauleColors](#)
- [interpretColor](#)
- [set](#)

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...  
scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► findLimitsAlgo ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_fanChart

Input:

- obj : An object of class nb_fanChart
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('central');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_plot. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_plot
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_plot with the given properties reset

Examples:

```
obj.set('propertyName',propertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ **nb_plotComb**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_plotComb(xData,yData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for plotting data with different plot types

The plotypes supported by this function are:

- 'area' : Uses the nb_area class
- 'line' : Uses the nb_plot class
- 'grouped' : Uses the nb_bar class, with the property style

```
        set to 'grouped'  
- 'stacked' : Uses the nb_bar class, with the property style  
        set to 'stacked'
```

Constructor:

```
obj = nb_plotComb(xData,yData,varargin)
```

Input:

- xData : The x-axis values of the plotted line. If only one input is given to this constructor it will be interpreted as the yData input. (Then the xData will be 1:size(yData,1))
- yData : The y-axis values of the plotted line. A double matrix. (1. dimension the number of observations and 2. dimension the number of variables.)

Optional input:

- varargin : 'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_plotComb (handle).

Examples:

```
nb_plotComb([0,2]);
```

same as

```
nb_plotComb([0,1],[0,2;2,3]);
```

```
nb_plotComb([0,1],[0,2],'propertyName',PropertyValue,...);
```

e.g.

```
nb_plotComb([0,1],[0,2;2,3],'types',{'area','line'},...  
'lineWidth',1.5);
```

Return the handle to the plotted line:

```
l = nb_plotComb([0,1],[0,2;2,3],'types',{'area','line'},...  
'lineWidth',1.5);
```

Which you can use to set and get properties:

```
l.set('cData',{'black','red'});  
lw = l.get('lineWidth');
```

Written by Kenneth SÃ¶terhagen Paulsen

Properties:

- abrupt
- alpha1
- alpha2
- barWidth
- baseValue
- baseline
- blend
- cData
- children
- clipping
- deleteOption
- direction
- edgeColor
- edgeLineStyle
- edgeLineWidth
- legendInfo
- lineWidth
- marker
- markerEdgeColor
- markerFaceColor
- markerSize
- parent
- shadeColor
- shaded
- side
- types
- visible
- xData
- yData

- **abrupt** ↑

Set this property to true to make the areas abruptly finish when given as nan.

- **alpha1** ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- **alpha2** ↑

Sets the alpha blending parameter nr 1 when blend is set to true.

- **barWidth** ↑

Width of the bars. As a scalar. 0.45 is default.

- **baseValue** ↑

The base value of the plot. Must be a scalar or a vector with size size(yData,1) x 1. The last option is not supported if any of the variables are plot as area!

- **baseline** ↑

A nb_horizontalLine handle (object). Use the set and get methods of this handle to change the baseline properties.

Caution: If baseValue is a vector this property is a nb_line object.

- **blend** ↑

When shaded is used, and this option is set to true, it will do alpha blending with shadeColor instead of shading.

- **cData** ↑

The color data of the line plotted. Must be of size; size(yData,2) x 3 with the RGB colors or a cellstr with size 1 x size(yData,2) with the color names.

- **children** ↑

The children of the handle. As nb_plot, nb_bar and nb_area handles (objects)

- **clipping** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **direction** ↑

The direction of the shading; {'north'} | 'south' | 'east' | 'west' (Only bar plot)

- **edgeColor** ↑

The color of the edge of the bars. In RGB colors | 'none' : No edgeLine | 'same' : Same color as the base color for each bar | a string with the color name.

- **edgeLineStyle** ↑

Line style of the edge of area and bar plots.

- **edgeLineWidth** ↑

Line width of the edge of area and bar plots.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-' | 'none'

- **lineWidth** [↑](#)

The line width of the plotted data. Must be a scalar.
Default is 2.5.

- **marker** [↑](#)

The markers of the lines. Either a string or a cell array with size as the number of lines to be plotted. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** [↑](#)

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerFaceColor** [↑](#)

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerSize** [↑](#)

Size of the markers. Must be an integer. Default is 9.

- **parent** [↑](#)

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **shadeColor** [↑](#)

The color the shaded bar plots are interpolated with.
A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **shaded** [↑](#)

Index of which data should be shaded. A matrix with size; size(yData,1) x 1 , 1 x size(yData,1) or size(yData,1) x number of variables to be plotted as bars

- **side** [↑](#)

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **types** ↑

A cell array of the plot type of each column of the property 'yData'. Must have same size as size(yData,2). Supported types : 'line','grouped','stacked' and 'area'.

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **xData** ↑

The x-axis data. As a double.

- **yData** ↑

The y-axis data. As a double.

Methods:

- [findClosestNiceNumber](#)
- [findLimitsAlgo](#)
- [get](#)
- [getDefauleColors](#)
- [interpretColor](#)
- [set](#)

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number,...  
scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_plotComb

Input:

- obj : An object of class nb_plotComb
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('baseline');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:**See also:**

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÃ¥','b'  
> 'red','rÃ¥d','r'  
> 'green','grÃ¶nn','g'  
> 'orange','o'  
> 'light blue','lys blÃ¥','lb'  
> 'black','svart','k'  
> 'grey','grÃ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÃ¥','hb'  
> 'deep blue','mÃ¥rk blÃ¥','db'  
> 'water blue','vannblÃ¥','vb'  
> 'cool grey','kald grÃ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_plotComb. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_plotComb
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_plotComb with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth SÃ¶therhagen Paulsen

■ nb_plotLabels

Go to: [Properties](#) | [Methods](#)

nb_annotation, handle

Description:

A class for plotting labels different types of plots.

Constructor:

```
obj = nb_plotLabels(varargin)
```

Input:

Optional input ('propertyName', PropertyValue) :

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_plotLabels

See also:

[nb_annotation](#), [handle](#), [nb_axes](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [children](#)
- [copyOption](#)
- [deleteOption](#)
- [fontName](#)
- [fontUnits](#)
- [formatAll](#)
- [formatCells](#)
- [formatColumns](#)
- [formatRows](#)
- [language](#)
- [listeners](#)
- [normalized](#)
- [parent](#)
- [pointer](#)
- [selected](#)

- **children** ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **fontName** ↑

Name of the font to use. Must be a string. Default is 'arial'.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **formatAll** ↑

Format of all the labels of this object. As 1 x 1 struct. Will be overwritten by formatColumns, formatRows and formatCells.

- **formatCells** ↑

Internal cell representation of each label. As a cell array of structs. Highest presedence.

- **formatColumns** ↑

Format of selected columns. As a cell array of structs. Will be overwritten by formatRows and formatCells.

- **formatRows** ↑

Format of selected rows. As a cell array of structs. Will be overwritten by formatCells.

- **language** ↑

Sets the language of the text. I.e. how decimal sign should be plotted. {'english'} | 'norwegian'.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object. When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

Methods:

- [set](#)
-

► [set](#) ↑

```
set(obj, varargin)
```

Description:

Set properties of the nb_plotLabels class

Input:

- varargin : Property name property value pairs.

Written by Kenneth S. Paulsen

■ nb_radar

Go to: [Properties](#) | [Methods](#)

```
obj = nb_radar(xData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a Class for making radar plots

This class will make the radar plot automatically fit the axes it is plot into.

I recommend to use the nb_* classes for the legend, titles and so on.

Constructor:

```
nb_radar(xData)
nb_radar(xData,'propertyName',propertyValue,...)
handle = nb_radar(xData,'propertyName',propertyValue,...)
```

Input:

- xData : The data to plot. size(xData,1) = number of variables and size(xData,2) = number of observations

Optional input:

- varargin : 'propertyName',propertyValue,...

Output:

- obj : An object of class nb_radar (handle).

Examples:

```
nb_radar(xData)
nb_radar(xData,'propertyName',PropertyValue,...)
handle = nb_radar(xData,'propertyName',PropertyValue,...)
radar = nb_radar([1,2; 2,4; 3,5; 2,1; 3,4; 2,5],...
    'numberOfIsoLines',10);
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- axesColor
- axesLimMax
- axesLimMin
- axesLineWidth
- cData
- deleteOption
- fontColor
- fontName
- fontSize
- fontUnits
- fontWeight
- isoLineColor
- isoLineStyle
- isoLineWidth
- labels
- legendInfo
- lineWidth
- location
- numberOfIsoLines
- parent
- rotate
- scale
- visible
- xData

- axesColor ↑

A 1x3 double with the RGB color or a string with the color name of the axes

- axesLimMax ↑

The axes max limit. Must be a double with size size(xData,1)x1

- axesLimMin ↑

The axes min limit. Must be a double with size size(xData,1)x1

- axesLineWidth ↑

The axes line width. As a scalar.

- cData ↑

The colors of the plotted data. Must be of size; length(xData) x 3. (Double with the RGB colors). Or a cellstr with the color names. Must be of size 1 x length(xData). See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **fontColor** ↑

The color of the font of the labels. Either a 1x3 double with the color of all the labels or a size(xData,1) x 3 with the color of each label (RGB colors). Can also be a string with the color names to use on all labels or a cellstr with the color names of each label. Size must be 1 x size(xData,1)

- **fontName** ↑

Name of the font used for the labels. As a string. 'arial' is default.

- **fontSize** ↑

Size of the font used for the labels. As a scalar.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

Weight of the font used for the labels. As a string. 'normal' is default.

- **isoLineColor** ↑

A 1x3 double with the RGB color or a string with the color name of the isocurves

- **isoLineStyle** ↑

The line style of the isocurves. As a string.

- **isoLineWidth** ↑

The line width of the isocurves. As a scalar.

- **labels** ↑

A cell array with the labels of the plot. Must have the same size as; size(xData,1). Each element of the cell array can consist of char with more lines. I.e.
{'A string',char('In a char','with two lines'),...}

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

Line style of the plotted data. As a string or a cellstr with size size(xData,2).

- **lineWidth** ↑

Line width of the plotted data. As a double.

- **location** ↑

'west' | 'east' | {'center'}. The location of the plot in the axes.

- **numberOfIsoLines** ↑

The number of isocurves of the radar plot. Must be an integer. Default is 10.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **rotate** ↑

Rotate the radar. In radians. Must be a scalar. Default is 0.

- **scale** ↑

Sets the axes limits, so the radar plot is scaled properly. (Also set the size of the plotted radar). Must be a 1x2 double. Default is [2,1.25].

- **visible** ↑

Sets the visibility of the radar plot. {'on'} | 'off'.

- **xData** ↑

The data to plot. size(xData,1) = number of observations and size(xData,2) = number of variables

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefauleColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                 scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_radar

Input:

- obj : An object of class nb_radar
- propertyName : A string with the propertyName

Output:

- PropertyValue : The value of the given property

Examples:

```
PropertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ;terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

set(obj,varargin)

Description:

Sets the properties of an object of class nb_radar. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_radar
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_radar with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_regressLine

Go to: [Properties](#) | [Methods](#)

nb_textAnnotation, nb_annotation, handle

Description:

A class for adding a regression line to a scatter plot.

Constructor:

```
obj = nb_regressLine(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_regressLine

Examples:

See also:

[nb_annotation](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- `cData`
- `copyOption`
- `fontName`
- `fontWeight`
- `interpreter`
- `listeners`
- `markerFaceColor`
- `parent`
- `positionY`
- `textBackgroundColor`
- `textLineWidth`
- `verticalAlignment`
- `children`
- `decimals`
- `fontSize`
- `horizontalAlignment`
- `lineStyle`
- `marker`
- `markerSize`
- `pointer`
- `selected`
- `textColor`
- `textMargin`
- `clipping`
- `deleteOption`
- `fontUnits`
- `index`
- `lineWidth`
- `markerEdgeColor`
- `normalized`
- `positionX`
- `string`
- `textEdgeColor`
- `textRotation`

- **`cData`** ↑

The color data of the line plotted. Must be of size; nRegLines x 3. With the RGB colors. Or a string with the name of the color. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- **`children`** ↑

The children of the handle.

Inherited from superclass `NB_ANNOTATION`

- **`clipping`** ↑

{'on'} | 'off' ; Clipping mode. MATLAB clips lines to the axes plot box by default. If you set Clipping to off, lines are displayed outside the axes plot box.

- **`copyOption`** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass `NB_ANNOTATION`

- **`decimals`** ↑

Number of decimals. Default is 2.

- **`deleteOption`** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass `NB_ANNOTATION`

- **fontName** ↑

Name of the font to use. Must be a string. Default is 'arial'.

Inherited from superclass NB_TEXTANNOTATION

- **fontSize** ↑

Sets the font size in the units given by the fontUnits property. Must be a scalar. Default is 12.

Inherited from superclass NB_TEXTANNOTATION

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

Inherited from superclass NB_TEXTANNOTATION

- **fontWeight** ↑

Weight of text characters. {'normal'} | 'bold' | 'light' | 'demi'

Inherited from superclass NB_TEXTANNOTATION

- **horizontalAlignment** ↑

Horizontal alignment of text. Specifies the horizontal justification of the text string. Must be a string.
'left' | 'center' | 'right'.

- **index** ↑

Selects the scatter plot to add the regression line to. I.e. if there is plotted to scatter plots to an axes this property can be set to either 1 or 2. (If <1, 1 is selected. If >2, 2 is selected. In this example)

- **interpreter** ↑

Interpret TeX instructions. Must be a string. 'latex' | {'tex'} | 'none'.

- **lineStyle** ↑

The line style of the plotted data. Must be a string.
{'-' } | '--' | ':' | '-' | 'none'. '---' is not supported.

- **lineWidth** ↑

The line width of the plotted data. As an integer.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **marker** ↑

The markers of the lines. Must be a string. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'}.

- **markerSize** ↑

Size of the markers. Must be a integer. Default is 9.

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TEXTANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object. When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **positionX** ↑

Manually set the x-axis position of the label. Must be a 1 x nRegLines double.

- **positionY** ↑

Manually set the x-axis position of the label. Must be a 1 x nRegLines double.

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **string** ↑

Text of regression line label. Default is the regression equation and R^{2} results. Must be a 1 x nRegLines cellstr.

- **textBackgroundColor** ↑

Color of text background rectangle. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names. Can also be set to 'none'.

- **textColor** ↑

Color of text. Must be of size; 1 x 3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- **textEdgeColor** ↑

Color of edge of text rectangle. Must be of size; 1 x 3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names. Can also be set to 'none'.

- **textLineWidth** ↑

Width of text rectangle edge. Must be a scalar. Default is 0.5.

- **textMargin** ↑

Space around text. A value in pixels that defines the space around the text string, but within the rectangle. Default value is 5 pixels.

- **textRotation** ↑

Text orientation. Determines the orientation of the text string. Specify values of rotation in degrees (positive angles cause counterclockwise rotation). Must be a scalar. Default is 0.

- **verticalAlignment** ↑

Vertical alignment of text. Specifies the vertical justification of the text string. Must be a string. {'top'} | 'cap' | 'middle' | 'baseline' | 'bottom'.

Methods:

- **set**

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_drawLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- `obj` : An object of class nb_drawLine
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class nb_drawLine with the given properties reset

Examples:

`obj.set('propertyName',propertyValue,...);`

Written by Kenneth Sæterhagen Paulsen

■ **nb_scatter**
Go to: [Properties](#) | [Methods](#)

`obj = nb_scatter()`

Superclasses:

handle, nb_plotHandle

Description:

Create scatter plot

Constructor:

obj = nb_scatter()

The constructor of the nb_scatter class

Input:

- xData : See below
- yData : See below

Optional inputs:

- varargin : ..., 'propertyName', PropertyValue, ...

Output:

- obj : An object of class nb_scatter

See also:

[scatter](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData
- children
- deleteOption
- legendInfo
- lineWidth
- marker
- markerSize
- parent
- side
- type
- visible
- xData
- yData

- **cData** ↑

Color of the plotted data. A matrix;
size(yData,2) x 3 (RGB color). Or a cellstr
with the color names. (With size 1 x size(yData,2))

- **children** ↑

The children of the object. As MATLAB scatter objects.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be
delete and all that is plotted by this object are removed
from the figure it is plotted on. If on the other hand
'only' is given, it will just delete the object.

- **legendInfo** ↑

{'off'} : Not included in the legend. 'on' : included in the legend.

- **lineStyle** ↑

Line style of the line connecting the markers. Default is 'none'.

- **lineWidth** ↑

Line width of line if plotted. Must be a scalar double.

- **marker** ↑

Sets the marker of the plot. Either {'o'} | 'x' | '+' | '*' | 's' | 'd' | '.' | '^' | '<' | '>' | 'v' | 'p' | 'h'. Must be a string or a cellstr with size 1 x size(yData,2).

- **markerSize** ↑

Sets the size of the markers. Must be a scalar.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is of class nb_axes. (Which is the default)

- **type** ↑

nb_scatter/type is a property.

- **visible** ↑

Sets the visibility of the radar plot. {'on'} | 'off'.

- **xData** ↑

A double matrix. Each column will be match against the corresponding column of th yData property. If it is given as a vector it will be expanded to fit the number of columns of the yData property.

- **yData** ↑

A double matrix. Each column will be match against the corresponding column of the xData property. If it is given as a vector it will be expanded to fit the number of columns of the xData property.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefauleColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get ↑**

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_scatter

Input:

- obj : An object of class nb_scatter
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('cData');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvit','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ;terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

set(obj,varargin)

Description:

Sets the properties of an object of class nb_scatter. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_scatter
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_scatter with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_table

Go to: [Properties](#) | [Methods](#)

This is a class for making tables.

Superclasses:

nb_plotHandle, nb_cursorTracker

Constructor:

```
obj = nb_table(data, varargin)
```

Input:

- data : Table cell contents as m x n cell array

Optional input:

- varargin : 'propertyName', PropertyValue,...

Output:

- obj : An object of class nb_table (handle).

Examples:

```
data = asCell(nb_cs.rand);
t = nb_table(data);
t.BackgroundColor(1, :) = {'blue'};
```

Written by Henrik Halvorsen Hortemo and Kenneth Sæterhagen Paulsen

Properties:

- [BackgroundColor](#)
- [BorderLeft](#)
- [Color](#)
- [DateFormat](#)
- [FontUnits](#)
- [Interpreter](#)
- [RowSpan](#)
- [allowDimensionChange](#)
- [decimals](#)
- [language](#)
- [size](#)
- [BorderBottom](#)
- [BorderRight](#)
- [ColumnSizes](#)
- [Editing](#)
- [FontWeight](#)
- [Margin](#)
- [String](#)
- [children](#)
- [defaultContextMenu](#)
- [mode](#)
- [stylingPatterns](#)
- [BorderColor](#)
- [BorderTop](#)
- [ColumnSpan](#)
- [FontSize](#)
- [HorizontalAlignment](#)
- [RowSizes](#)
- [VerticalAlignment](#)
- [data](#)
- [deleteOption](#)
- [parent](#)
- [updateOnChange](#)

- **[BackgroundColor](#)** ↑

m x n cell array. Cell contents: [R G B] | color name as string

- **[BorderBottom](#)** ↑

Line width of bottom border
m x n cell array. Cell contents: Integer

- **[BorderColor](#)** ↑

[R G B] or color name

- **[BorderLeft](#)** ↑

Line width of left border
m x n cell array. Cell contents: Integer

- **[BorderRight](#)** ↑

Line width of right border
m x n cell array. Cell contents: Integer

- **[BorderTop](#)** ↑

Line width of top border
m x n cell array. Cell contents: Integer

- **[Color](#)** ↑

m x n cell array. Cell contents: [R G B] | color name as string

- **[ColumnSizes](#)** ↑

Normalized 1 x n vector

- **ColumnSpan** ↑

Sets how many columns a cell should span.
Cells expand downwards and to the right.
m x n cell array. Cell contents: Integer

- **DateFormat** ↑

m x n cell array. Cell contents: char

- **Editing** ↑

m x n cell array. Cell contents: logical

- **FontSize** ↑

m x n cell array. Cell contents: double

- **FontUnits** ↑

m x n cell array. Cell contents: string

- **FontWeight** ↑

m x n cell array. Cell contents: 'bold' | 'normal'

- **HorizontalAlignment** ↑

m x n cell array. Cell contents: 'left' | 'center' | 'right'

- **Interpreter** ↑

m x n cell array. Cell contents: 'none' | 'tex' | 'latex'

- **Margin** ↑

Margin from left/right border to text
m x n cell array. Cell contents: Integer

- **RowSizes** ↑

Normalized 1 x m vector

- **RowSpan** ↑

Sets how many rows a cell should span
Cells expand downwards and to the right.
m x n cell array. Cell contents: Integer

- **String** ↑

m x n cell array. Cell contents: cellstr

- **VerticalAlignment** ↑

m x n cell array. Cell contents: 'left' | 'center' | 'right'

- **allowDimensionChange** ↑

Boolean indicating whether or not adding/removing rows/columns is allowed

- **children** ↑

Cell array of object handles

- **data** ↑

A cell of the unformatted data of the table.

- **decimals** ↑

Round numbers to the number of digits. Default is not to round numbers. Either as an integer with the number of digits, or a string with the printed format. See num2str function.

- **defaultContextMenu** ↑

The default context menu to add to each cell. Add, Delete and Format menus will be appended. If empty only these three menus are added.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **language** ↑

Sets the language of the date format given by the DateFormat property. Only important for the Monthtext and Quartertext options.

- **mode** ↑

Which type of edit mode the table is in. Either 'even' (default) or 'neighbour'. This affect how the rest of the rows or columns of the table is resizes given the change in one of the row or column. If 'even' all the rest of the rows or columns of the table share the adjustment in the row or column evenly, otherwise the next row column in the line take all the adjustment.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **size** ↑

[m n]

- **stylingPatterns** ↑

Styling patterns. See nb_table.stylingPatternsTemplate method.

- **updateOnChange** ↑

Turn automatic updates on/off. Used for performance enhancements and to avoid recursion traps

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefaultColors](#)
 - [interpretColor](#)
 - [set](#)
 - [struct](#)
 - [unstruct](#)
-

► **findClosestNiceNumber** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scal factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **findLimitsAlgo** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow,dataHigh,method,fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylimit : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get** ↑

```
propertyValue = get(obj, propertyName)
```

Description:

Get a property of an object of class nb_table

Input:

- obj : An object of class nb_table
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors ↑**

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the nb_plotHandle class.

Get the default colors given the number of plotted variables.

Input:

- dataSize : Number of plotted variables.

Output:

- defaultColors : The default colors, as a M x 3 double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the nb_plotHandle class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blÅ¥','b'  
> 'red','rÅ,d','r'  
> 'green','grÅ,nn','g'  
> 'orange','o'  
> 'light blue','lys blÅ¥','lb'  
> 'black','svart','k'  
> 'grey','grÅ¥','gr'  
> 'purple','lilla','p'  
> 'yellow','gul','y'  
> 'magenta','m'  
> 'white','hvít','w'  
> 'cyan','c'  
> 'burgundy','burgunder'  
> 'sky blue','himmelblÅ¥','hb'  
> 'deep blue','mÅ,rk blÅ¥','db'  
> 'water blue','vannblÅ¥','vb'  
> 'cool grey','kald grÅ¥','cgr'  
> 'sand','s'  
> 'pink','rosa','pi'  
> 'turquoise','turkis','t'
```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÅ!terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set** ↑

```
set(obj, varargin)
```

Description:

Set properties of the nb_table class

Input:

- varargin : Property name property value pairs.

Written by Henrik Halvorsen Hortemo

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct

Input:

- obj : An object of class nb_table

Output:

- s : A struct

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_table.unstruct(s)

Description:

Convert struct to object

Input:

- s : A struct

Output:

- obj : An object of class nb_table

See also:

[nb_table.struct](#)

■ nb_tableTextUpdateEvent

Go to: [Properties](#) | [Methods](#)

The data that get sent when the updatedTableText event is notified.

Superclasses:

event.EventData

See also:

[nb_table](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [EventName](#)
- [Index](#)
- [Source](#)
- [String](#)

- [EventName](#) ↑

event.EventData.EventName - Name of event

Documentation for event.EventData.EventName
doc event.EventData.EventName

- [Index](#) ↑

The index of the edited cell of the nb_table object.

- [Source](#) ↑

event.EventData.Source - Event source

Documentation for event.EventData.Source
doc event.EventData.Source

- [String](#) ↑

A char with the string property of the edited cell of the nb_table object

Methods:

■ nb_textArrow

Go to: [Properties](#) | [Methods](#)

`nb_lineAnnotation, nb_movableAnnotation, nb_textAnnotation,
nb_annotation, handle`

Description:

A class for adding a arrow with a text box to a plot.

Constructor:

`obj = nb_textArrow(varargin)`

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- `obj` : An object of class `nb_textArrow`

Examples:

See also:

`nb_textAnnotation, nb_annotation, handle, nb_graph_ts
nb_graph_cs`

Written by Kenneth Sætherhagen Paulsen

Properties:

- `children`
- `color`
- `copyOption`
- `deleteOption`
- `fontAngle`
- `fontName`
- `fontSize`
- `fontUnits`
- `fontWeight`
- `headLength`
- `headStyle`
- `headWidth`
- `horizontalAlignment`
- `interpreter`
- `lineStyle`
- `lineWidth`
- `listeners`
- `normalized`
- `parent`
- `pointer`
- `selected`
- `side`
- `string`
- `textBackgroundColor`
- `textColor`
- `textEdgeColor`
- `textLineWidth`
- `textMargin`
- `textRotation`
- `units`
- `verticalAlignment`
- `xData`
- `yData`

- **`children`** ↑

The children of the handle.

Inherited from superclass `NB_ANNOTATION`

- **color** ↑

The color data of the line plotted. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass `NB_ANNOTATION`

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass `NB_ANNOTATION`

- **fontAngle** ↑

{'normal'} | 'italic' | 'oblique'

- **fontName** ↑

Name of the font to use. Must be a string. Default is 'arial'.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontSize** ↑

Sets the font size in the units given by the `fontUnits` property. Must be a scalar. Default is 12.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

normalized - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

Inherited from superclass `NB_TEXTANNOTATION`

- **fontWeight** ↑

Weight of text characters. {'normal' | 'bold' | 'light'
| 'demi'

Inherited from superclass NB_TEXTANNOTATION

- **headLength** [↑](#)

Length of first arrowhead. Size in points. As a scalar.
Specify in points.1 point = 1/72 inch. The default value
is 8.

The first arrowhead is located at the start defined by
the point xData(1), yData(1).

- **headStyle** [↑](#)

Style of first arrowhead. As a string. Specify this
property as one of the strings from the following table.

```
> 'none'  
> 'plain'  
> 'ellipse'  
> 'vback1'  
> 'vback2' (default)  
> 'vback3'  
> 'cback1'  
> 'cback2'  
> 'cback3'  
> 'star4'  
> 'rectangle'  
> 'diamond'  
> 'rose'  
> 'hypocycloid'  
> 'astroid'  
> 'deltoid'
```

- **headWidth** [↑](#)

Width of first arrowhead. Specify in points.
1 point = 1/72 inch. As a scalar. The default value is
8.

The arrowhead is located at the start defined by the
point xData(1), yData(1).

- **horizontalAlignment** [↑](#)

Horizontal alignment of text. Specifies the horizontal
justification of the text string. Must be a string.
'left' | 'center' | 'right' | {'default'}. When 'default'
is given this class will try to find the best
alignment.

- **interpreter** [↑](#)

Interpret TeX instructions. Must be a string. 'latex' |
{'tex'} | 'none'.

- **lineStyle** ↑

The line style of the plotted data. Must be a string.
{'-' } | '--' | ':' | '-' | 'none'. '---' is not supported.

- **lineWidth** ↑

The line width of the plotted arrow. As an integer.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TEXTANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object. When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **selected** ↑

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** ↑

The axes to plot against. 'left' or 'right'. 'left' is default.

- **string** ↑

Sets the text of the textbox. Must be a char. Each vertical element of the given char will result in a new line of text.

- **textBackgroundColor** ↑

Color of text background rectangle. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names. Can also be set to 'none'.

- **textColor** ↑

Color of text. Must be of size; 1 x 3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names.

- **textEdgeColor** ↑

Color of edge of text rectangle. Must be of size; 1 x 3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names. Can also be set to 'none'.

- **textLineWidth** ↑

Width of text rectangle edge. Must be a scalar. Default is 0.5.

- **textMargin** ↑

Space around text. A value in pixels that defines the space around the text string, but within the rectangle. Default value is 5 pixels.

- **textRotation** ↑

Text orientation. Determines the orientation of the text string. Specify values of rotation in degrees (positive angles cause counterclockwise rotation). Must be a scalar. Default is 0.

- **units** ↑

The coordinates system to plot in. Either {'data'} : xy-coordinates | 'normalized' : figure coordinates.

- **verticalAlignment** ↑

Vertical alignment of text. Specifies the vertical justification of the text string. Must be a string. 'top' | 'cap' | 'middle' | 'baseline' | 'bottom' | 'default'. When 'default' is given this class will try to find the best alignment.

- **xData** ↑

The x-axis data. As a double. E.g: [x_begin, x_end].

- **yData** ↑

The y-axis data. As a double. E.g: [y_begin, y_end].

Methods:

- [set](#)

► **set** ↑

`set(obj,varargin)`

Description:

Sets the properties of an object of class nb_textArrow. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- `obj` : An object of class nb_textArrow
- `varargin` : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- `obj` : The same object of class nb_arrow with the given properties reset

Examples:

`obj.set('propertyName',PropertyValue,...);`

Written by Kenneth Sæterhagen Paulsen

■ **nb_textBox**

Go to: [Properties](#) | [Methods](#)

`nb_textAnnotation, nb_movableAnnotation, nb_lineAnnotation,
nb_annotation, handle`

Description:

A class for adding a text box to a plot.

Constructor:

```
obj = nb_textBox(varargin)
```

Input:

Optional input ('propertyName', PropertyValue):

> You can set some properties of the class. See the list below.

Output

- obj : An object of class nb_textBox

Examples:

See also:

[nb_textAnnotation](#), [nb_lineAnnotation](#), [nb_annotation](#), handle
[nb_graph_ts](#), [nb_graph_cs](#), [nb_graph_data](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- | | | | |
|-----------------------------------|-------------------------------------|------------------------------|---------------------------------------|
| • backgroundColor | • children | • color | • copyOption |
| • deleteOption | • edgeColor | • fontAngle | • fontName |
| • fontSize | • fontUnits | • fontWeight | • horizontalAlignment |
| • interpreter | • lineStyle | • lineWidth | • listeners |
| • margin | • normalized | • parent | • pointer |
| • rotation | • selected | • side | • string |
| • units | • verticalAlignment | • xData | • yData |

- **backgroundColor** ↑

Color of text background rectangle. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names. Can also be set to 'none'.

- **children** ↑

The children of the handle.

Inherited from superclass NB_ANNOTATION

- **color** ↑

Color of text. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- **copyOption** ↑

Set to true if a copy option should be added to the context menu of annotation object. Default is false.

Inherited from superclass NB_ANNOTATION

- **deleteOption** ↑

{'all'} | 'only'. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object

Inherited from superclass NB_ANNOTATION

- **edgeColor** ↑

Color of edge of text rectangle. Must be of size; 1x3. With the RGB colors. Or a string with the name of the color. See the method interpretColor of the nb_plotHandle class for more on the supported color names. Can also be set to 'none'.

- **fontAngle** ↑

{'normal'} | 'italic' | 'oblique'

- **fontName** ↑

Name of the font to use. Must be a string. Default is 'arial'.

Inherited from superclass NB_TEXTANNOTATION

- **fontSize** ↑

Sets the font size in the units given by the font units property. Must be a scalar. Default is 12.

Inherited from superclass NB_TEXTANNOTATION

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

Inherited from superclass NB_TEXTANNOTATION

- **fontWeight** ↑

Weight of text characters. {'normal'} | 'bold' | 'light'
| 'demi'

Inherited from superclass NB_TEXTANNOTATION

- **horizontalAlignment** ↑

Horizontal alignment of text. Specifies the horizontal justification of the text string. Must be a string.
{'left'} | 'center' | 'right'.

- **interpreter** ↑

Interpret TeX instructions. Must be a string.
'latex' | {'tex'} | 'none'.

- **lineStyle** ↑

The line style of the plotted box. Must be a string.
{'-' } | '--' | ':' | '-' | 'none'. '---' is not supported.

- **lineWidth** ↑

Width of text rectangle edge. Must be a scalar. Default is 0.5.

- **listeners** ↑

The listeners of the handle.

Inherited from superclass NB_ANNOTATION

- **margin** ↑

Space around text. A value in pixels that defines the space around the text string, but within the rectangle. Default value is 5 pixels.

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

Inherited from superclass NB_TEXTANNOTATION

- **parent** ↑

The parent as an nb_axes object

Inherited from superclass NB_ANNOTATION

- **pointer** ↑

The pointer selected when the mouse is above the object.
When not above the object '' is selected.

Inherited from superclass NB_ANNOTATION

- **rotation** [↑](#)

Text orientation. Determines the orientation of the text string. Specify values of rotation in degrees (positive angles cause counterclockwise rotation). Must be a scalar. Default is 0.

- **selected** [↑](#)

Indicates if the object is being selected.

Inherited from superclass NB_ANNOTATION

- **side** [↑](#)

The axes to plot against. 'left' or 'right'. 'left' is default.

- **string** [↑](#)

Sets the text of the textbox. Must be a char. Each vertical element of the given char will result in a new line of text.

- **units** [↑](#)

The coordinates system to plot in. {'data'} : xy-coordinates. 'normalized' : Axes are normalized to have length 1.

- **verticalAlignment** [↑](#)

Vertical alignment of text. Specifies the vertical justification of the text string. Must be a string. 'top' | 'cap' | {'middle'} | 'baseline' | 'bottom'.

- **xData** [↑](#)

The x-axis data. As a scalar.

- **yData** [↑](#)

The y-axis data. As a scalar.

Methods:

- [set](#)

► **set** [↑](#)

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_textArrow. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_textArrow
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_arrow with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_title

Go to: [Properties](#) | [Methods](#)

```
obj = nb_title(parent,string,varargin)
```

Superclasses:

handle

Description:

This is a class for adding a title to a nb_axes handle

Constructor:

```
obj = nb_title(parent,string,varargin)
```

Input:

- parent : An object of class nb_axes
- string : The title as a string

Optional input:

- varargin : ...,'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_title

Examples:

First you need to make the nb_axes handle
ax = nb_axes();

Then you can add the title
nb_title(ax,'A title');

nb_title(ax,'A title','propertyName',PropertyValue,...);
titleHandle = nb_title(ax,'A title','propertyName',...
PropertyValue,...);

same as

nb_title('A title','parent',ax,'propertyName',...
PropertyValue,...);

titleHandle = nb_title('A title','parent',ax,...
'propertyName',PropertyValue,...);

Make new axes for the given title:

nb_title('A title','propertyName',PropertyValue,...);
titleHandle = nb_title('A title','propertyName',...
PropertyValue,...);

See also:

[title](#), [nb_axes](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- alignment
- children
- deleteOption
- fontName
- fontSize
- fontUnits
- fontWeight
- interpreter
- normalized
- parent
- placement
- string
- visible

- **alignment** ↑

The figure title text alignment, default is 'center'. You also have 'left' and 'right'.

- **children** ↑

The children will consist of one MATLAB title handle.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **fontName** ↑

The font name to use. As a string. Default is 'arial'

- **fontSize** ↑

The size of the font. As a scalar.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the `fontSize` property.

normalized - Interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `fontSize` accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The weight of the font, {'bold'} | 'normal' | 'light'.

- **interpreter** ↑

{'none'} | 'tex' | 'latex'

- **normalized** ↑

Indicate if the font when `fontUnits` is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

- **parent** ↑

The parent, as a nb_axes handle (object).

- **placement** ↑

The placement of the title. {'center'} | 'left' | 'right' | 'lefttaxes'

- **string** ↑

The text of the title. As a char. Multi-lined char gives a multi-lined title.

- **visible** ↑

Sets the visibility of the radar plot. {'on'} | 'off'.

Methods:

- **get**
 - **set**
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_fanLegend

Input:

- **obj** : An object of class nb_fanLegend
- **propertyName** : A string with the propertyName

Output:

- **propertyValue** : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_title. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_title
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_title with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_verticalLine

Go to: [Properties](#) | [Methods](#)

```
obj = nb_verticalLine(xData,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for making vertical lines which spans all of the y-axis.

If the parent is a nb_axes object:

- The line will automatically update the length of the line, so it spans all the y-axis.

If the parent is a MATLAB axes handle (object):

- The line will not update itself automatically and you need to use the update() method of the object to make it fit to the axes handle (If it is changed)

Constructor:

```
obj = nb_verticalLine(xData,varargin)
```

Input:

- xData : The x-axis value for where to place the vertical line. As a scalar.

Optional input:

- varargin : ...,'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_verticalLine

Examples:

```
nb_verticalLine(xData)
obj = nb_verticalLine(xData)
obj = nb_verticalLine(xData,varargin)
```

See also:

[nb_line](#), [line](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- cData
- children
- deleteOption
- legendInfo
- lineWidth
- marker
- markerEdgeColor
- markerFaceColor
- parent
- side
- visible
- xData
- **lineStyle**
- **markerSize**

- **cData** [↑](#)

The color data of the line plotted. Must be of size; 1x3. With the RGB colors. or a string with the color name. See the method `interpretColor` of the `nb_plotHandle` class for more on the supported color names.

- **children** [↑](#)

The children of this object. As an `nb_line` object.

- **deleteOption** [↑](#)

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **legendInfo** [↑](#)

{'off'} : Not included in the legend. 'on' : included in the legend

- **lineStyle** ↑

The line style of the plotted data. Either a string or a cell array with size as the number of lines to be plotted. {'-' } | '--' | '---' | ':' | '-' | 'none'

- **lineWidth** ↑

The line width of the plotted data. Must be a scalar.
Default is 1.

- **marker** ↑

The markers of the lines. Either a string or a cell array with size as the number of lines to be plotted. See marker property of the MATLAB line function for more on the options of this property

- **markerEdgeColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerFaceColor** ↑

A 1x3 double with the RGB colors | 'none' | {'auto'} | a string with the color name.

- **markerSize** ↑

Size of the markers. Must be a integer. Default is 9.

- **parent** ↑

The parent axes you wan to plot on, if not given it will be plotted in a new nb_axes handle. Either a axes or a nb_axes handle (recommended).

Inherited from superclass NB_PLOTHANDLE

- **side** ↑

{'left'} | 'right' ; Which axes to plot on. Only if the parent is off class nb_axes. (Which is the default)

- **visible** ↑

Sets the visibility of the plotted lines. {'on'} | 'off'.

- **xData** ↑

A scalar with the x-axis value for where to place the vertical line.

Methods:

- [findClosestNiceNumber](#)
 - [findLimitsAlgo](#)
 - [get](#)
 - [getDefauleColors](#)
 - [interpretColor](#)
 - [set](#)
-

► **[findClosestNiceNumber](#)** ↑

```
niceNumber = nb_plotHandle.findClosestNiceNumber(number, ...
                                                scaleNumber, up)
```

Description:

A static method of the nb_plotHandle class.

Find first "nice" number, used for deciding the y-axis limits when findAxisMethod == 3 is used.

Input:

- number : The given number to find a "nice" number close to.
- scaleNumber : The scale factor used. I.e. the precision of the algorithm.
- up : 1 if rounding up, 0 if rounding down.

Output:

- niceNumber : The number which is "nice" and close to the input number.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **[findLimitsAlgo](#)** ↑

```
ylimit = nb_plotHandle.findLimitsAlgo(dataLow, dataHigh, method, fig)
```

Description:

A static method of the nb_plotHandle class.

Figure out the limits of prepared data by the subclasses of this class

Input:

- dataLow : A double with the lowest values to plot
- dataHigh : A double with the highest values to plot
- method : The method to use when finding the y-axis limits.
 - > 1: Uses the floor and ceil methods. (Give problem when graphing small number)
 - > 2: Add some space in both direction of the y-axis. (The limits will seldom be nice numbers.)
 - > 3: Uses the submethod findClosestNiceNumber of the class to decide on the limits. (Also this method have can have problems with data which is not symmetric around the zero line.)
 - > 4 : Uses the MATLAB algorithm for finding the axis limits.

Output:

- ylim : The y-axis limits. Given by a 1 x 2 double.
[lowerlimit upperlimit]

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **get ↑**

propertyValue = get(obj,propertyName)

Description:

Get a property of an object of class nb_verticalLine

Input:

- obj : An object of class nb_verticalLine
- propertyName : A string with the propertyName

Output:

- `propertyValue` : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **getDefaultColors** ↑

```
defaultColors = nb_plotHandle.getDefaultColors(dataSize)
```

Description:

A static method of the `nb_plotHandle` class.

Get the default colors given the number of plotted variables.

Input:

- `dataSize` : Number of plotted variables.

Output:

- `defaultColors` : The default colors, as a $M \times 3$ double of the RGB colors. M is the number of plotted variables.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_PLOTHANDLE`

► **interpretColor** ↑

```
colors = nb_plotHandle.interpretColor(colors)
```

Description:

A static method of the `nb_plotHandle` class.

Interpret the colors when given as a cell array of strings

Supported color names are:

```
> 'blue','blå','b'
```

```

> 'red','rÃ,d','r'
> 'green','grÃ,nn','g'
> 'orange','o'
> 'light blue','lys blÃ¥','lb'
> 'black','svart','k'
> 'grey','grÃ¥','gr'
> 'purple','lilla','p'
> 'yellow','gul','y'
> 'magenta','m'
> 'white','hvit','w'
> 'cyan','c'
> 'burgundy','burgunder'
> 'sky blue','himmelblÃ¥','hb'
> 'deep blue','mÃ,rk blÃ¥','db'
> 'water blue','vannblÃ¥','vb'
> 'cool grey','kald grÃ¥','cgr'
> 'sand','s'
> 'pink','rosa','pi'
> 'turquoise','turkis','t'

```

Input:

- colors : A cellstr with the colornames. E.g. {'red','green'}

Output:

- colors : A M x 3 double with the RGB colors matching the input cellstr. Where M is the length of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_PLOTHANDLE

► **set ↑**

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_verticalLine. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_verticalLine
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_verticalLine with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_xlabel

Go to: [Properties](#) | [Methods](#)

```
obj = nb_xlabel(parent,string,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for adding a x-label to a nb_axes handle

Constructor:

```
obj = nb_xlabel(parent,string,varargin)
```

Input:

- parent : An object of class nb_axes
- string : the label as string

Optional input:

- varargin : 'propertyName',PropertyValue,...

Output:

- obj : A handle to the plot.

Examples:

First you need to make the nb_axes handle
ax = nb_axes();

Then you can add the x-label
nb_xlabel(ax,'A label');

Other examples:

```
nb_xlabel(ax,'A label','propertyName',PropertyValue,...);
```

```
titleHandle = nb_xlabel(ax,'A label','propertyName',...
                           PropertyValue,...);
```

same as

```
nb_xlabel('A label','parent',ax,'propertyName',...
    'propertyValue,...');

xlabelHandle = nb_xlabel('A label','parent',ax,...
    'propertyName',propertyValue,...);
```

Make new axes for the given x-label:

```
nb_xlabel('A label','propertyName',propertyValue,...);

xlabelHandle = nb_xlabel('A label','propertyName',...
    'propertyValue,...');
```

See also

`nb_axes`, `xlabel`

Written by Kenneth Sætherhagen Paulsen

Properties:

- `alignment`
- `children`
- `deleteOption`
- `extent`
- `fontName`
- `fontSize`
- `fontUnits`
- `fontWeight`
- `interpreter`
- `normalized`
- `offset`
- `parent`
- `placement`
- `string`
- `visible`

- **alignment** ↑

The figure title text alignment, default is 'center'. You also have 'left' and 'right'.

- **children** ↑

The children will consist of one MATLAB text handle.

- **deleteOption** ↑

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **extent** ↑

position rectangle (read-only)

Position and size of text. A four-element vector that defines the size and position of the text string:

[left,bottom,width,height]

left and bottom are the x- and y-coordinates of the lower left corner of the text extent. The units are normalized.

- **fontName** ↑

The font name to use. As a string. 'arial' is default.

- **fontSize** ↑

The size of the font. As a scalar. Default is 14.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' |
'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The weight of the font, {'bold'} | 'normal' | 'light'

- **interpreter** ↑

{'none'} | 'tex' | 'latex'.

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

- **offset** ↑

A scalar which manage the space between the x-axis tick mark labels and the x-label.

- **parent** ↑

The parent, as a nb_axes handle (object).

- **placement** ↑

The placement of the title. {'center'} | 'left' | 'right' | 'leftaxes'

- **string** ↑

The text of the x-label. As a string or a char.

- **visible** ↑

{'on'} | 'off'.

Methods:

- [get](#)
 - [set](#)
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_xlabel

Input:

- obj : An object of class nb_xlabel
- propertyName : A string with the propertyName

Output:

- propertyValue : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_xlabel. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_xlabel
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_xlabel with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_ylabel

Go to: [Properties](#) | [Methods](#)

```
obj = nb_ylabel(parent,string,varargin)
```

Superclasses:

handle, nb_plotHandle

Description:

This is a class for adding a y-label to a nb_axes handle

Constructor:

```
obj = nb_ylabel(parent,string,varargin)
```

Input:

- parent : An object of class nb_axes
- string : The y-axis label as a string

Optional input:

- varargin : 'propertyName',PropertyValue,...

Output:

- obj : An object of class nb_ylabel

Examples:

First you need to make the nb_axes handle
ax = nb_axes();

```
Then you can add the y-label  
nb_ylabel(ax,'A label');
```

Other examples:

```
nb_ylabel(ax,'A label','propertyName',PropertyValue,...);
```

```
titleHandle = nb_ylabel(ax,'A label','propertyName',...  
                           PropertyValue,...);
```

same as

```
nb_ylabel('A label','parent',ax,'propertyName',...  
                           PropertyValue,...);
```

```
ylabelHandle = nb_ylabel('A label','parent',ax,...  
                           'propertyName',PropertyValue,...);
```

Make new axes for the given y-label:

```
nb_ylabel('A label','propertyName',PropertyValue,...);
```

```
ylabelHandle = nb_ylabel('A label','propertyName',...  
                           PropertyValue,...);
```

See also:

[nb_axes](#), [ylabel](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [children](#)
- [deleteOption](#)
- [extent](#)
- [fontName](#)
- [fontSize](#)
- [fontUnits](#)
- [fontWeight](#)
- [interpreter](#)
- [normalized](#)
- [offset](#)
- [parent](#)
- [side](#)
- [string](#)
- [visible](#)

- **children** [↑](#)

The children will consist of one MATLAB text handle.

- **deleteOption** [↑](#)

'all' | {'only'}. If 'all' is given the object will be delete and all that is plotted by this object are removed from the figure it is plotted on. If on the other hand 'only' is given, it will just delete the object.

- **extent** [↑](#)

position rectangle (read-only)

Position and size of text. A four-element vector that defines the size and position of the text string:

[left,bottom,width,height]
left and bottom are the x- and y-coordinates of the lower left corner of the text extent. The units are normalized.

- **fontName** ↑

The font name to use. As a string. 'arial' is default.

- **fontSize** ↑

The size of the font. As a scalar. Default is 14.

- **fontUnits** ↑

{'points'} | 'normalized' | 'inches' |
'centimeters' | 'pixels'

Font size units. MATLAB uses this property to determine the units used by the fontSize property.

normalized - Interpret FontSize as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen fontSize accordingly.

pixels, inches, centimeters, and points:
Absolute units. 1 point = 1/72 inch.

- **fontWeight** ↑

The weight of the font, {'bold'} | 'normal' | 'light'.

- **interpreter** ↑

{'none'} | 'tex' | 'latex'.

- **normalized** ↑

Indicate if the font when fontUnits is set to 'normalized' should be normalized to the axes ('axes') or to the figure ('figure'), i.e. the default axes position [0.1 0.1 0.8 0.8].

- **offset** ↑

A number which manage the space between the y-axis tick mark labels and the y-axis label. Default is 0.

- **parent** ↑

The parent, as a nb_axes handle (object).

- **side** ↑

The side of the y-label. As a string. Either 'left' or 'right'. 'left' is default.

- **string** ↑

The text of the y-label. As a string or a char.

- **visible** ↑

{'on'} | 'off'.

Methods:

- **get**
 - **set**
-

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property of an object of class nb_ylabel

Input:

- **obj** : An object of class nb_ylabel
- **propertyName** : A string with the propertyName

Output:

- **propertyValue** : The value of the given property

Examples:

```
propertyValue = obj.get('children');
```

Written by Kenneth S. Paulsen

► **set** ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_ylabel. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties could be set with one method call.

Input:

- obj : An object of class nb_ylabel
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_ylabel with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

■ nb_editArrow

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_arrow interactively.

Constructor:

```
gui = nb_editArrow(parent)
```

Input:

- parent : An object of class nb_arrow.

Output:

- gui : An object of class nb_editArrow

See also:

[nb_arrow](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- defaultColors • figureHandle • parent
- [defaultColors ↑](#)

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_drawPatch object

Methods:

■ **nb_editBarAnnotation**
Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_barAnnotation interactively.

Constructor:

gui = nb_editBarAnnotation(parent)

Input:

- parent : An object of class nb_barAnnotation.

Output:

- gui : An object of class nb_editBarAnnotation.

See also:

[nb_barAnnotation](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- defaultColors • figureHandle • parent

- **defaultColors** ↑

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_barAnnotation object

Methods:

■ **nb_editColorBar**

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_colorbar interactively.

Constructor:

gui = nb_editColorBar(parent)

Input:

- parent : An object of class nb_colorbar.

Output:

- gui : An object of class nb_editColorBar.

See also:

[nb_colorbar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

• defaultColors • figureHandle • parent

- **defaultColors** ↑

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_textBox object

Methods:

■ **nb_editCurlyBrace**

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_curlyBrace interactively.

Constructor:

gui = nb_editCurlyBrace(parent)

Input:

- parent : An object of class nb_curlyBrace.

Output:

- gui : An object of class nb_editCurlyBrace

See also:

[nb_arrow](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- defaultColors • figureHandle • parent

- [defaultColors ↑](#)

Default colors

- [figureHandle ↑](#)

A handle to the GUI window

- [parent ↑](#)

As an nb_drawPatch object

Methods:

■ [nb_editDrawLine](#)

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_editDrawLine interactively.

Constructor:

gui = nb_editDrawLine(parent)

Input:

- parent : An object of class nb_drawLine.

Output:

- gui : An object of class nb_editDrawLine

See also:

[nb_drawLine](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- defaultColors • figureHandle • parent

- [defaultColors ↑](#)

Default colors

- **figureHandle** ↑
A handle to the GUI window

- **parent** ↑
As an nb_drawPatch object

Methods:

■ **nb_editDrawPatch**
Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_drawPatch interactivly.

Constructor:

gui = nb_editDrawPatch(parent)

Input:

- parent : An object of class nb_drawPatch.

Output:

- gui : An object of class nb_editDrawPatch

See also:

[nb_drawPatch](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

• defaultColors • figureHandle • parent

- **defaultColors** ↑

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_drawPatch object

Methods:

■ **nb_editPlotLabels**
Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_plotLabels interactively.

Constructor:

gui = nb_editPlotLabels(parent, class, index)

Input:

- parent : An object of class nb_barAnnotation.

Output:

- gui : An object of class nb_editPlotLabels.

See also:

[nb_barAnnotation](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

• class • defaultColors • figureHandle • index • parent

• type

- **class** ↑

Class for the underlying object to add labels to. Some options may be affected by this.

- **defaultColors** ↑

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **index** ↑

The index of the label that is being edited.

- **parent** ↑

As an nb_barAnnotation object

- **type** ↑

The level of editing. Either 'all', 'row', 'column' or 'cell'

Methods:

■ **nb_editRegressLine**

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_regressLine interactivly.

Constructor:

gui = nb_editRegressLine(parent)

Input:

- parent : An object of class nb_regressLine.

Output:

- gui : An object of class nb_editRegressLine

See also:

[nb_regressLine](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

• defaultColors • figureHandle • parent

- **defaultColors** ↑

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_drawPatch object

Methods:

■ **nb_editTextArrow**

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_textArrow interactivly.

Constructor:

gui = nb_editTextArrow(parent)

Input:

- parent : An object of class nb_textArrow.

Output:

- gui : An object of class nb_editTextArrow.

See also:

[nb_textArrow](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- defaultColors • figureHandle • parent

- [defaultColors ↑](#)

Default colors

- [figureHandle ↑](#)

A handle to the GUI window

- [parent ↑](#)

As an nb_textBox object

Methods:

■ [nb_editTextBox](#)

Go to: [Properties](#) | [Methods](#)

A class for editing an object of class nb_textBox interactively.

Constructor:

```
gui = nb_editTextBox(parent)
```

Input:

- parent : An object of class nb_textBox.

Output:

- gui : An object of class nb_editTextBox.

See also:

[nb_textBox](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- defaultColors • figureHandle • parent

- [defaultColors ↑](#)

Default colors

- **figureHandle** ↑

A handle to the GUI window

- **parent** ↑

As an nb_textBox object

Methods:

◆ **nb_alpha**

c = nb_alpha(c1,c2,alpha1,alpha2)

Description:

Alpha blending of two colors.

Input:

- c1 : A 1x3 double with the RGB color.
- c2 : A 1x3 double with the RGB color.
- alpha1 : A 1x1 double between 0 and 1.
- alpha2 : A 1x1 double between 0 and 1.

Output:

- c : A 1x3 double with the alpha blended RGB color.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_breakText**

nb_breakText(textObj, fitTo)

Description:

Wrap text of a object given some area to fit to. The object is already assumed to be included in the area fitTo, and therefore only the extent of the text is checked to not cross the right border.

Input:

- textObj : Handle to a text object
- fitTo : A 1x4 double with position to fit the text to.

Written by Kenneth Sæterhagen Paulsen and Henrik Halvorsen Hortemo

◆ nb_breakTextAfterEdit

```
index = nb_breakTextAfterEdit(textObj, fitTo, index)
```

Description:

Break text of a object given some area to fit to, after editing. I.e. after one string element is added to the string property of the object. The object is already assumed to be included in the area fitTo, and therefore only the extent of the text is checked to not cross the right border.

Input:

- textObj : Handle to a text object
- fitTo : A 1x4 double with position to fit the text to.
- index : A 1x2 double with the index for where the string property of the text handle is breaked. index(1) is the current row, and index(2) is the current column. Will be updated if the breakpoint is before the current position in a line. I.e. the current point must then be switched to the next line

Written by Kenneth Sæterhagen Paulsen

◆ nb_colormap

```
nb_colormap(ax, map)
```

Description:

Set the color map used by an object of class nb_axes. The color map of an nb_axes object may be used when a nb_image object assign the nb_axes object as it parent.

Input:

- ax : An object of class nb_axes.
- map : A n x 3 double with the color map to use.

See also:

[nb_axes.colorMap](#), [nb_axes.set](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_defaultColors

```
defaultColors = nb_defaultColors()
defaultColors = nb_defaultColors(number,isNB)
```

Description:

Gets the default colors used by the NB toolbox.

To change the default colors set the environment variable 'defaultColors' to the name of a .m file (fully specified path!) that creates a variable defaultColors. This variable must have size N x 3, and be a double array.

Example;

```
setenv('C:\yourDefaultColors.m')
```

Where the file is on the format;

```
defaultColors = [
    0, 0, 0;
    0.8, 0.8, 0.8
];
```

Output:

defaultColors : A n x 3 double with the RGB colors.

Written by Kenneth Sæterhagen Paulsen

◆ nb_displayValue

```
nb_displayValue(hObject, event)
```

Description:

Callback function which can be used to display the value of the data point in a nb_notifiesMouseOverObject object.

Input:

- hObject : An object of class nb_axes.
- event : A nb_mouseOverObjectEvent object
- variables : A nested cellstr with the same size as the maximum value of event.h (Number of children of the axes). And each element must have size event.y (Number of plotted variables by the child).
- variablesX : A nested cellstr with the same size as the maximum value of event.h (Number of children of the axes). And each element must have size event.y (Number of plotted observation by the child).

Output:

- A text box on screen displaying the data value.

Example:

```
d      = rand(10,2);
d(2,2) = -0.2;
d(2,1) = -0.1;
bar    = nb_bar(d,'style','stacked');
func   = @(src,event) nb_displayValue(src,event,{{'Var1','Var2'}});
addlistener(bar.parent,'mouseOverObject',func);
```

See also:

[nb_notifiesMouseOverObject](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_dpos2dpos**

```
x = nb_dpos2dpos(x,oldPos,newPos,oldScale,newScale)
```

Description:

Convert the matrix *x* from the old units to the new units value.

Input:

- *x* : Old value(s). As a double matrix.
- *oldPos* : Old positions. As a 1x2 double.
- *newPos* : New positions. As a 1x2 double.
- *oldScale* : New scale. Either 'normal' or 'log'.
- *newScale* : Old scale. Either 'normal' or 'log'.

Output:

- *x*

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getCurrentPointInAxesUnits**

```
[cPoint,cAxPoint] = nb_getCurrentPointInAxesUnits(fig,ax)
```

Description:

Get current point of the mouse in the axes units of an nb_axes object.

Input:

- fig : Either a figure, a nb_figure or a nb_graphPanel object.
- ax : As a nb_axes or axes object.

Output:

- cPoint : Current point in figure or panel units. Will depend on if the fig input is an nb_figure or an nb_graphPanel object.
- cAxPoint : Current point of the mouse in the axes units of an nb_axes object. (in normalized units).

Written by Kenneth Sæterhagen Paulsen

◆ nb_getFanColors

```
cData = nb_getFanColors(cData,percentiles)
```

Description:

Get fan colors

Input:

- cData : A string with the base color to use. {'nb'} | 'yellow' | 'magenta' | 'cyan' | 'red' | 'green' | 'blue' | 'grey' | 'white'
- percentiles : The percentiles as 1 x numberofPercentiles double. E.g: [.3,.5,.7,.9] (Which is default)

Output:

- cData : A numberofPercentiles x 3 double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_getFontSize

```
fontSize = nb_getFontSize(fontSize,newFontUnits,oldFontUnits,...  
                           normalizeFactor)
```

Description:

Transform the given font size from a font units from a old to a new one while keeping the font size on the plot. (See below for an exception)

Caution : If you transform from 'points','inches' and

'centimeters' to 'normalized' this function will not keep the font size, it will normalize the font size to how the given font size would have been plotted on a axes with positions [0.13 0.11 0.775 0.815] on a screen with resolution 1680x1050 pixels.

Adjust the normalizeFactor to change which axes positions and screen resolution to normalize against.

Input:

- fontSize : The font size to convert. As a scalar.
- newFontUnits : The font units to convert to.
Either 'points', 'inches', 'centimeters' or 'normalized'.
- oldFontUnits : The font to convert from.
Either 'points', 'inches', 'centimeters' or 'normalized'.
- normalizeFactor : The factor that transform the font size in 'points' to the font size in 'normalized' units. The default factor is 0.001787310098302, which will normalize the font size to how the given font size would have been plotted on a axes with positions [0.13 0.11 0.775 0.815] on a screen with resolution 1680x1050 pixels.

Output:

- fontSize : The converted font size. As a scalar.

Examples:

```
fontSize = nb_getFontSize(12,'normalized','points');
```

See also:

Written by Kenneth Sæterhagen Paulsen

◆ **nb_htmlColors**

```
colors = nb_htmlColors(endc)
```

Description:

Create a color list for use in colored pop-up menus

Input:

- endc : A n x 3 double with the RGB colors.

Output:

- colors : A cell array with size n.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_interpretKeyPress**

```
[index,special] = nb_interpretKeyPress(textObj,index,event)
```

Description:

Update a text object given a index and a event triggered by KeyReleaseFcn
KeyPressFcn or a nb_keyEvent.

Input:

- textObj : A MATLAB text handle (object).
- index : A 1x2 double with the index for where the string property of
the text handle is edited. index(1) is the current row, and
index(2) is the current column.
- event : Either a struct thrown by KeyReleaseFcn or KeyPressFcn, or a
nb_keyEvent.

Output:

- index : The updated index. May change due to arrow being pressed.
- special : Either:
 - '' : No special key is pressed.
 - 'escape' : Esc is pressed.
 - 'tab' : Tab is pressed.
 - struct : If the special field is set to
'multilinePaste', a multilined element
is pasted into the given location. See
the pasted field for the pasted element.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_isAxes**

```
ret = nb_isAxes(value)
```

Description:

Test if an object is of class axes.

Input:

- value : Any

Output:

- ret : True if value is a handle to a axes.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isColorProp

```
ret = nb_isColorProp(in,log)
```

Description:

Evaluates the color properties. Must have dimension size N x 3 with the RGB colors or a cellstr with size 1 x N with the color names.

N == 1 if mult is set to false, else N can be > 1.

Input:

- in : The property value to be evaluated.

- mult : Allow for multiple colors.

Output:

- ret : Logical. Returns true if the property is evaluated to be a valid color property as stated in the description above.

Written by Tobias Ingebrigtsen

◆ nb_isContextMenu

```
ret = nb_isContextMenu(value)
```

Description:

Test if an object is of class ContextMenu.

Input:

- value : Any

Output:

- ret : True if value is a handle to a figure.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isFigure

```
ret = nb_isFigure(value)
```

Description:

Test if an object is of class figure.

Input:

- value : Any

Output:

- ret : True if value is a handle to a figure.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isPanel

```
ret = nb_isPanel(value)
```

Description:

Test if an object is of class panel.

Input:

- value : Any

Output:

- ret : True if value is a handle to a figure.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isRGB

```
ret = nb_isRGB(in)
ret = nb_isRGB(in,moreRows)
```

Description:

Test if the input in is a RGB color.

Input:

```
- in      : The value to be evaluated.  
- moreRows : Give true to also tolerate a N x 3 RGB color matrix for  
N > 1. Default is false.
```

Output:

```
- ret      : Logical. Returns true if the input is a RGB color.
```

Written by Tobias Ingebrigtsen

◆ **nb_isUipanel**

```
ret = nb_isUipanel(value)
```

Description:

Test if an object is of class ContextMenu.

Input:

```
- value : Any
```

Output:

```
- ret    : True if value is a handle to a figure.
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_islineStyle**

```
ret = nb_islineStyle(in,allowCell)
```

Description:

Test if an input is a valid lineStyle property.

Input:

```
- in      : Any input
```

```
- allowCell : True if we allow for cellstring.
```

Output:

```
- ret : true if in is a valid lineStyle property.
```

Written by Tobias Ingebrigtsen

◆ nb_ismarker

```
ret = nb_ismarker(in)
```

Description:

Test if an input is a valid marker property.

Input:

- in : Any input.
- log : True if we allow for cellstr.

Output:

- ret : true if in is a valid marker property.

Written by Tobias Ingebrigtsen

◆ nb_istype

```
ret = nb_istype(in)
```

Description:

Test if an input is a valid type property.

Input:

- in : Any input

Output:

- ret : true if in is a valid type property.

Written by Tobias Ingebrigtsen

◆ nb_pos2pos

```
x = nb_pos2pos(x,oldPos,newPos,oldScale,newScale)
```

Description:

Convert the value x from the old units to the new units value.

Input:

```
- x           : Old value. As a double.  
- oldPos     : Old positions. As a 1x2 double.  
- newPos     : New positions. As a 1x2 double.  
- oldScale   : New scale. Either 'normal' or 'log'.  
- newScale   : Old scale. Either 'normal' or 'log'.
```

Output:

```
- x
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_scaleLineWidth**

```
lineWidth = nb_scaleLineWidth(obj, lineWidth)
```

Description:

Scale a line width property.

Input:

```
- obj        : An object of class nb_plotHandle  
- lineWidth : The value of a property that represent the width of some  
               type of line. As a double.
```

Output:

```
- lineWidth : The scaled or non-scaled line width.
```

See also:

[nb_axes.scaleLineWidth](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_setDefinedColorAnnotation**

```
nb_setDefinedColorAnnotation(hObject, event, gui, popupmenu)
```

Description:

Make the user able to define a color and add it to the popupmenu handle.

Input:

- hObject : The MATLAB handle triggering the event
- event : The event structure.
- gui : An gui objects thots edit the nb_annotation objects. Must have a property defaultColors.
- varargin : Handle to the popupmenu handle to assign the defined color. Can be given more handles as seperate inputs

Written by Kenneth Sætherhagen Paulsen

◆ nb_subplot

```
nb_axesObject = nb_subplot(m,n,p,varargin)
```

Description:

Create axes in tiled positions.

obj = nb_subplot(m,n,p), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot, and returns the nb_axes handle. The axes are counted along the top row of the Figure window, then the second row, etc.

Input:

- m : Number of axes rows of current figure
- n : Number of axes columns of current figure
- p : The p-th axes of the subplot
- varargin : ..., 'propertyName', 'PropertyValue', ... combination given to the initialized nb_axes handles. Should not include the property name 'position', because then you overrule the position given by this function. See the nb_axes class for more on the properties to set.

Caution : If you want to add more sub axes to one figure it is important to provide the 'parent' property. E.g. if you want two axes in one figure, do the following:

```
fig = nb_figure;
axes1 = nb_subplot(2,1,1,'parent',fig);
axes2 = nb_subplot(2,1,2,'parent',fig);
```

Output:

- nb_axesObject : An object of class nb_axes

Examples:

```
nb_axesObject = nb_subplot(2,2,1);
nb_axesObject = nb_subplot(2,2,1,'shading','grey');
```

See also:

[nb_axes](#), [axes](#), [subplot](#), [nb_subplotSpecial](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_subplotSpecial**

```
nb_axesObject = nb_subplotSpecial(m,n,p,varargin)
```

Description:

Create axes in tiled positions. Contrary to the `nb_subplot` function this function will tile the `1x2` and `2x1` subplot in positions better fitted for inclusion in presentations.

`obj = nb_subplotSpecial(m,n,p)`, breaks the Figure window into an `m`-by-`n` matrix of small axes, selects the `p`-th axes for the current plot, and returns the `nb_axes` handle. The axes are counted along the top row of the Figure window, then the second row, etc.

Input:

- `m` : Number of axes rows of current figure
- `n` : Number of axes columns of current figure
- `p` : The `p`-th axes of the subplot
- `varargin` : ..., 'propertyName', `PropertyValue`, ... combination given to the initialized `nb_axes` handles. Should not include the property name 'position', because then you overrule the position given by this function. See the `nb_axes` class for more on the properties to set.

Caution : If you want to add more sub axes to one figure it is important to provide the 'parent' property. E.g. if you want two axes in one figure, do the following:

```
fig = nb_figure;
axes1 = nb_subplotSpecial(2,1,1,'parent',fig);
axes2 = nb_subplotSpecial(2,1,2,'parent',fig);
```

Output:

- `nb_axesObject` : An object of class `nb_axes`

Examples:

```
nb_axesObject = nb_subplotSpecial(2,2,1);
nb_axesObject = nb_subplotSpecial(2,2,1,'shading','grey');
```

See also:

[nb_axes](#), [axes](#), [subplot](#), [nb_subplot](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_subplot_position**

```
position = nb_subplot_position(m,n,p)
```

Description:

Get the axes position when divided into subplots.

`obj = nb_subplot(m,n,p)`, breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot, and returns the nb_axes handle. The axes are counted along the top row of the Figure window, then the second row, etc.

Input:

- m : Number of axes rows of current figure
- n : Number of axes columns of current figure
- p : The p-th axes of the subplot

Output:

- position : A 1x4 double. [xLow,yLow,xWidth,yWidth].

Examples:

```
position = nb_subplot(2,2,1);
```

See also:

[subplot](#), [nb_subplot](#), [nb_subplotSpecial](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_unitsRatio**

Calculate ratio between different units

Examples:

```
nb_unitsRatio('Characters', 'Normalized', axes);
```

Written by Henrik Halvorsen Hortemo

◆ **nb_wrapFigureText**

```
nb_wrapFigureText(obj,t,place)
```

Description:

Wrap footer or figure title.

Input:

- obj : An object of class nb_figure or nb_figureTitle.
- t : A text handle of the displayed footer or figure title.
- place : x-axis position of where the text is positioned.

See also:

[nb_footer](#), [nb_figureTitle](#)

Written by Kenneth Sæterhagen Paulsen

25.5 Utils graphics classes and functions

- [nb_getInUnits](#)
- [nb_gobjects](#)
- [nb_oldGraphVersion](#)

◆ **nb_getInUnits**

```
position = nb_getInUnits(object, property, units)
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_gobjects**

```
array = nb_gobjects(varargin)
```

Description:

Preallocate array for storing graphical objects that are robust to different versions of MATLAB.

Input:

- See gobjects or nan.

Output:

- array : Either a matrix of nan or a matrix of default graphics objects. See gobjects or nan. If nb_oldGraphVersion returns true then doubles are returned.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_oldGraphVersion**

```
ret = nb_oldGraphVersion()
```

Description:

MATLAB handle graphical objects in a new way since version 8.4. This return true if the version the user is using is less than this.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

25.6 Cell functions

- RemoveDuplicates
- nb_appendIndexes
- cell2charTable
- nb_cell2latex
- nb_appendLagPostfix
- nb_cell2String
- nb_cellstr2file
- nb_cellfind
- nb_cellstr2String
- nb_cellstrlead
- nb_compareCell2TXT
- nb_cellstrlag
- nb_convertContexts
- nb_nestedCell2Cell
- nb_contains
- nb_scellstrlag
- nb_strcomb
- nb_obj2cell

◆ **RemoveDuplicates**

```
out = RemoveDuplicates(in,dim)
```

Description:

Remove duplicates of a sorted cellstr array (also a matrix)

Input:

- in : A cellstr which are sorted in the dimesnion to remove the duplicates.
- dim : The dimesnion to remove the duplicates.

Output:

- out : A cellstr with the duplicates removed.

Written by Junior Maih

◆ **cell2charTable**

```
tableAsChar = cell2charTable(table)
```

Description:

Convert a cell matrix to a printable char vector.

Input:

- table : A cellstr matrix.

Output:

- tableAsChar : A char vector.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_appendIndexes**

```
c = nb_appendIndexes(str,indexes)
```

Description:

Append indexes to a string, and return those strin in an cellstr array.

Input:

- str : A one line char (string).

- indexes : The index to append. E.g. 1:10.

Output:

- c : A cellstr.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_appendLagPostfix**

```
newNames = nb_appendLagPostfix(c,nLags)
```

Description:

Append '_lagX' postfix to a cellstr.

Input:

- nLags : A cell. E.g. {1,2,4,8}
- c : A cellstr.

Output:

- xout : A cellstr

See also:

[nb_cellstrlag](#)

Written by Kenneth S. Paulsen

◆ **nb_cell2String**

string = nb_cell2String(c)

Description:

Convert a cell to a string. Each element will be called with the `toString` method.

Input:

- c : A cell.

Output:

- string : A string.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_cell2latex**

code = nb_cell2latex(c,varargin)

Description:

Make latex table of one or more cellstr matrices. Please see the preamble for the packages needed to make the latex code work.

Input:

- c : A cellstr matrix.

Optional input:

- 'caption' : Adds a caption to the table. Must be a string.
If the 'tables' input is given this input must be a cell array with same length as that input. (or empty)
- 'colors' : Set it to 0 if the colored rows of the table should be turned off. Default is 1.
- 'columnOrientation' : A cellstr with size 1 x size(c,2). Each element must be either 'l', 'c', 'r', 'R{Xcm}', 'C{Xcm}' or 'R{Xcm}'.
If the 'tables' input is given this input must be a cell array with same length as that input. (or empty)
- 'firstRowColor' : A string with the color, e.g. 'blue','white','black','blue!20!white'. Default is 'nbblue'.
- 'fontSize' : Sets the font size of the table. Either:
'Huge', 'huge', 'LARGE', 'Large', 'large',
'normalsize' (default), 'small',
'footnotesize', 'scriptsize','tiny'.

Caution : This option is case sensitive.
- 'justification' : Sets the justification of the caption of the table. Must be a string.
Either 'justified','centering' (default),
'raggedright', 'RaggedRight', 'raggedleft'.

Caution : This option is case sensitive.
- 'language' : Either 'english' (default) or 'norwegian'
('norsk').
- 'lookUpMatrix' :

Sets how the given mnemonics (variable names) should map to different languages. Must be cell array on the form:


```
{'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```


Or a .m file name, as a string. The .m file must include (and only include) what follows:


```
lookUpMatrix = {  
  
'mnemonics1','englishDescription1','norwegianDescription1';
 'mnemonics2','englishDescription2','norwegianDescription2'};
```

Be aware : Each of the given description can be multi-lined char, which will result in multi-line text of the table.

- 'preamble' : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.
- 'rotation' : The rotation of the table in LaTeX. Either:
 - > 'sidewaystable' : A sideways table
 - > 'landscape' : Rotate the whole page
 - > '' : No rotation
- 'rowColor1' : Color of every second row of the table starting from the second row. Same options as was the case for the 'firstRowColor' option.
- 'rowColor2' : Color of every second row of the table starting from the third row. Same options as was the case for the 'firstRowColor' option.
- 'lineBreak' : The maximum number of characters before line break. The line breaking take place at white spaces. Default is no line break.
- 'maxRows' : Maximum number of rows of a table in one page. If c has more rows than this number the table will start over again in the next page. Default is [], i.e. tables will not be broken even if not fit to one page.
- 'space' : Number of lines between multiple tables. Default is 5.
- 'tables' : A cell array with more cellstr matrices to produce latex tables of. Important for nice breaking of many huge cell matrices. The c input will be discarded in this case.

Output:

- code : A char with the code.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_cellfind**

```
ind = nb_cellfind(c,pattern)
```

Description:

Check if the given pattern is found in the cells of the cellstr array. Will return a logical array of same size. Return true for

all cells where the pattern is found.

Input:

- c : A cellstr array
- pattern : The pattern to look for. As a string

Output:

- ind : Logical array

Examples:

```
c = {'dfsdf','dfdf.df','sdfs'};  
ind = nb_cellfind(c,'.')
```

Written by Kenneth S. Paulsen

◆ nb_cellstr2String

```
str = nb_cellstr2String(c)  
str = nb_cellstr2String(c,sep,sepLast)
```

Description:

Convert a cellstr array to a string using the sep input to separate the elements of the cellstr.

Input:

- c : A cellstr.
- sep : A string. Default is ','
- sepLast : A string. Default is ','

Output:

- str : A string.

Example:

```
c = {'dfdfddf','dfdffdf','dfdfdf'};  
str = nb_cellstr2String(c,', ','')
```

Gives str = 'dfdfddf, dfdffdf, dfdfdf'

Written by Kenneth Sætherhagen Paulsen

◆ nb_cellstr2file

```
nb_cellstr2file(string,filename)
nb_cellstr2file(string,filename,breakLine)
```

Description:

Write a cellstr to a file.

Input:

- string : A cellstr or char array.
- filename : The name of the saved file. If the extension is not provided the default is '.m'.
Can also be a file identifier.
- breakLine : true || false. Add '\r\n' at the end of each line or not.
Default is true.

Written by Kenneth SÃ¶therhagen Paulsen

◆ **nb_cellstrlag**

```
cflag = nb_cellstrlag(c,nlag,type)
```

Description:

Type = 'lagFast':

Creates a array of size 1 x size(c,2)*nlag. I.e.
{'var1_lag1', 'var1_lag2', ... 'var1_lagN', ...
'var2_lag1', ... 'var2_lagN'}

Type = 'varFast':

Creates a matrix of size 1 x size(c,2)*nlag. I.e.
['var1_lag1', ... 'varp_lag1', ..., 'var1_lagN', ...
'varp_lagN']

Input:

- c : A cellstr
- nlag : The number of lags. Default is 1. Could be a 1 x nvar double.
- type : Either 'lagFast' (default) or 'varFast'

Output:

- xout : A cellstr

Written by Kenneth S. Paulsen

◆ nb_cellstrlead

```
clag = nb_cellstrlead(c,nlead,type,reverse)

Description:
Type = 'leadFast':
Creates a matrix of size(x,1) x size(x,2)*nlead. I.e.
{'var1_lead1', 'var1_lead2', ... 'var1_leadN', ...
 'var2_lead1', ... 'var2_leadN'}
Type = 'varFast':
Creates a matrix of size 1 x size(c,2)*nlag. I.e.
['var1_lead1', ... 'varp_lead1', ..., 'var1_leadN', ...
 'varp_leadN']
```

Input:

- c : A cellstr
- nlead : The number of lead. Default is 1. Could be a 1 x nvar double.
- type : Either 'leadFast' (default) or 'varFast'

Output:

- xout : A cellstr

Written by Kenneth S. Paulsen

◆ nb_COMPARECell2TXT

```
nb_COMPARECell2TXT(cell,file)
```

Description:

Check to see if the cell is equal to the contents of the .TXT file. This function assumes that the .TXT file has be written by nb_cellstr2file.

Input:

- cell : A cellstr.
- filename : The name of the saved file with the full path.
- breakLine : true || false. Add '\r\n' at the end of each line or not. Default is true.

Ouput:

- bool : Logical. True if contents (and formatting) is equal. False if they are different. Will also return false if file cannot be opened / found.

Written by Per Bjarne Bye

◆ nb_contains

```
ind = nb_contains(s,pattern,varargin)
```

Description:

Robust implementation of the MATLAB function contains over different versions of MATLAB.

Input:

- s : A M x N cellstr or a M x 1 char.
- pattern : A one line char with the pattern to look for.

Optional input:

- 'ignoreCase' : Set to true to ignore case.

Output:

- ind : A M x N logical if input is cellstr, otherwise M x 1. true if pattern found, otherwise false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_convertContexts

```
contexts = nb_convertContexts(contexts)
```

Description:

Convert from a cellstr of elements on the format 'yyyymmdd' to a double vector.

Input:

- contexts : Either a 1 x N or N x 1 cellstr, where all the elements are on the format 'yyyymmdd'.

Output:

- contexts : A N x 1 double vector.

Written by Kenneth Sæterhagen Paulsen

◆ nb_nestedCell2Cell

c = nb_nestedCell2Cell(nested)

Description:

Transform a nested cell array to a cell array

Input:

- nested : A nested cell array. E.g. {{'Var1','Var2'},{'Var3'}}

Output:

- c : A cell array. E.g. {'Var1','Var2','Var3'}

Written by Kenneth Sæterhagen Paulsen

◆ nb_obj2cell

cout = nb_obj2cell(obj)

Description:

Convert object matrix to cell matrix

Input:

- obj : An user defined object of size M x N x P

Output:

- c : A cell of size M x N x P

Written by Kenneth Sæterhagen Paulsen

◆ nb_scellstrlag

clag = nb_scellstrlag(c, lags)

Description:

```
Creates a array of nObs x total number of lags. I.e.  
[x1(t-1), x1(t-2), ... x1(t-max([lags{1}]), x2(t-1), ... ,  
x2(t-max([lags{1}]))]
```

Input:

- c : A cellstr
- nlag : The number of lags. Default is 1. Could be a 1 x nvar double.
- type : Either 'lagFast' (default) or 'varFast'

Output:

- xout : A cellstr

Examples:

```
cflag = nb_scellstrlag({'Var1','Var2'}, {1,[1,2]})
```

Written by Kenneth S. Paulsen

◆ **nb_strcomb**

```
c = nb_strcomb(c1,c2)  
c = nb_strcomb(c1,c2,sep)
```

Description:

Create all combinations of two cellstr.

Input:

- c1 : A cellstr with size N x 1
- c2 : A cellstr with size M x 1
- sep : A string with the separator between the two concatenated strings. Default is '_'.

Output:

- c : A cellstr with size N*M

Examples:

```
c1 = {'T';'G';'F'};  
c2 = {'a';'b';'c'};  
c = nb_strcomb(c1,c2)
```

See also:

Written by Kenneth Sæterhagen Paulsen

25.7 Char functions

- nb_any2String
 - nb_breakChar
 - nb_createChar
 - nb_dash
 - nb_getMatchingParentheses
 - nb_isOneLineChar
 - nb_newLine
 - nb_splitIntoTerms
 - nb_str2double
 - toString
-

◆ nb_any2String

```
string = toString(anyObject)
```

Description:

Same as `toString`, but `cellstr` and one line char are converted to the way they are displayed by MATLAB instead of using `nb_cellstr2String`.

Input:

- `anyObject` : A double, logical, `nb_ts`, `nb_cs`, `nb_data` or cell.

Output:

- `string` : A string

See also:

`num2str`, `int2str`, `nb_cellstr2String`, `toString`

Written by Kenneth Sæterhagen Paulsen

◆ nb_breakChar

```
s = nb_breakChar(s,numChar)
```

Description:

Break char so that `size(s,2) <= numChar`

Input:

- `s` : A char.
- `numChar` : An integer > 0 .

Output:

- `s` : A char.

Written by Kenneth Sæterhagen Paulsen

◆ nb_createChar

```
out = nb_createChar(r,c)
```

Description:

Create a empty r x c char array.

Input:

- r : Number of rows.
- c : Number of columns.

Output:

- out : A r x c char array.

Written by Kenneth Sæterhagen Paulsen

◆ nb_dash

```
d = nb_dash(type,printing2PDF)
```

Description:

Get dash type.

Input:

- type : Either 'dash', 'en-dash' or 'em-dash'. Default is 'dash'.

Output:

- d : A char with the given dash type.

Written by Kenneth Sæterhagen Paulsen

◆ nb_getMatchingParentheses

```
match = nb_getMatchingParentheses(expr,reverse)
```

Description:

Find index of matching parentheses.

Input:

- expr : A one-line char that starts with (.
- reverse : Set to true to match in from end to start. Default is false.

Output:

- match : A 1 x 2 integer with the index of the matching parentheses in expr. Will be given as [0,0], if not located.

Examples:

```
match = nb_getMatchingParentheses(' (y*x+(1-y)*x) ')
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_isOneLineChar**

```
ret = nb_isOneLineChar(str)
```

Description:

Test if an input is a one line (row) char.

Input:

- in : Any input

Output:

- ret : true if in is a one line char, otherwise false.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_newLine**

```
nl = nb_newLine
nl = nb_newLine(num)
```

Description:

Returns the new line character, same as char(10). Same as newline, but added to be robust to different versions of MATLAB. The num input is also added compared to newline.

Input:

- num : The number of new line characters to make. Default is 1.

Output:

- nl : A one line char with the new line characters.

See also:

[newline](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_splitIntoTerms

```
breaks          = nb_splitIntoTerms(expr)
[breaks,split,addPar] = nb_splitIntoTerms(expr,op,notOp)
```

Description:

Split a mathematical expression into separate terms.

Input:

- expr : A one-line char that include a mathematical expression.
- op : Operators to split at. Default is {'+', '-'} . Must be a cellstr.
- notOp : Operators that will invalidates break-points if just in front of any operator in op. Default is {'*', '^', '/'}. Must be a cellstr.

Output:

- breaks : The indexes of '+' and '-' signs.
- split : A nBreaks + 1 cell array.
- addPar : The expression should be enclosed with parentheses again.

Examples:

```
[breaks,split,addPar] = nb_splitIntoTerms('y*x+(1-y)*x')
[breaks,split,addPar] = nb_splitIntoTerms(' (y*x+(1-y)*x)')
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_str2double

```
num = nb_str2double(string)
```

Description:

Convert a string to double. Added the possibility to convert a interval from a string to a double compared to str2double. See examples

Input:

- string : A string.

Output:

- num : A double

Examples:

```
nb_str2double('2')
nb_str2double('1:3')
nb_str2double('[1,3]')
nb_str2double('[1:3,6]')
```

See also:

[str2double](#)

Written by Kenneth Sætherhagen Paulsen

◆ **toString**

```
string = toString(anyObject)
```

Description:

Converts object into a string.

Input:

- anyObject : A double, logical, nb_ts, nb_cs, nb_data or cell.

Output:

- string : A string

See also:

[num2str](#), [int2str](#), [nb_cellstr2String](#), [nb_any2String](#)

Written by Kenneth Sætherhagen Paulsen

25.8 Help Classes

- [nb_logicalInExpr](#)

■ **nb_logicalInExpr**
Go to: [Properties](#) | [Methods](#)

A class representing a logical array that can be used in nb_eval to get the resulting logical array from an expression.

Constructor:

obj = nb_logicalInExpr(data)

Input:

- data : Logical array.

Output:

- obj : An object of class nb_logicalInExpr.

Examples:

```
obj = nb_logicalInExpr(true);
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- data • type

- **data** ↑

The date the object represents.

- **type** ↑

The type of logical operator to use. Default is '||' (or).

Methods:

• abs	• acos	• acosd	• acosh	• acot
• acotd	• acoth	• acsc	• acscd	•acsch
• and	• append	• asec	• asecd	• asech
• asin	• asind	• asinh	• atan	• atand
• atanh	• avgOAF	• bkfilter	• bkfilter1s	• callfun
• ceil	• conj	• convert	• corr	• cos
• cosd	• cosh	• cot	• cotd	• coth
• cov	• csc	• cscd	• csch	• cumprod
• cumsum	• demean	• demeanMoving	• denton	• deptcat
• detrend	• diff	• disp	• egrowth	• epcn
• eq	• exp	• expand	• expm1	• extrapolate
• fillNaN	• floor	• ge	• growth	• gt
• h2l	• horzcat	• horzcatfast	• hpfilter	• hpfilter1s
• iegrowth	• iegrowthnan	• iepcn	• iepcnnan	• igrowth
• igrowthnan	• interpolate	• ipcn	• ipcnnan	• isfinite
• isnan	• kurtosis	• lag	• ldivide	• le
• lead	• log	• log10	• log1p	• log2
• lt	• mavg	• max	• mean	• median
• merge2Series	• min	• minus	• mldivide	• mpower
• mrdivide	• mstd	• mtimes	• ne	• not
• or	• pca	• pca_func	• pcn	• plus
• pow2	• power	• q2y	• rdivide	• reIndex
• real	• reallog	• realpow	• realsqrt	• ret
• rlag	• round	• sec	• secd	• sech
• set2Value	• setNan2Zero	• sgrowth	• sin	• sind
• sinh	• skewness	• sqrt	• std	• stdise
• subAvg	• subSum	• sum	• sumOAF	• tan
• tand	• tanh	• times	• uminus	• undiff
• uplus	• var	• vertcat	• window	• x12
• x12Census				

► **abs** ↑

```
obj = abs(obj)
```

Description:

Absolute value

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = abs(in);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acos** ↑

```
obj = acos(obj)
```

Description:

Take inverse cosine, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acos(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acosd** ↑

```
obj = acosd(obj)
```

Description:

acosd(obj) is the inverse cosine, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acosd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acosh** ↑

```
obj = acosh(obj)
```

Description:

acosh(obj) is the inverse hyperbolic cosine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acosh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acot** ↑

```
obj = acot(obj)
```

Description:

Take inverse cotangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acotd** ↑

```
obj = acotd(obj)
```

Description:

acotd(obj) is the inverse cotangent, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acotd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acoth** ↑

```
obj = acoth(obj)
```

Description:

acoth(obj) is the inverse hyperbolic cotangent of the elements of obj

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acoth(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsc** ↑

```
obj = acsc(obj)
```

Description:

Take acsc

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = acsc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsqd** ↑

```
obj = acscd(obj)
```

Description:

acscd(obj) is the inverse cosecant, expressed in degrees, of the elements of obj

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acscd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **acsch** ↑

```
obj = acsch(obj)
```

Description:

acsch(obj) is the inverse hyperbolic cosecant

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = acsch(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **and** ↑

```
a = and(a,b)
```

Description:

The and operator (&).

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr

Output:

- a : An object of class nb_objectInExpr.

Examples:

```
a = a & b;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **append ↑**

```
obj = append(obj,aObj)
obj = append(obj,aObj,priority)
```

Description:

Append aObj to obj with a given priority.

Input:

- obj : A nb_objectInExpr object.
- aObj : A nb_objectInExpr object.
- priority : 'first' or 'second'. If 'first' all the observations from obj is used before the observations from aObj, otherwise it is reverse. 'first' is default.

Output:

- obj : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asec** ↑

```
obj = asec(obj)
```

Description:

Inverse secant, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = asec(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asecd** ↑

```
obj = asecd(obj)
```

Description:

asecd(obj) is the inverse secant, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asecd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asech** ↑

```
obj = asech(obj)
```

Description:

asech(obj) is the inverse hyperbolic secant

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asech(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asin** ↑

```
obj = asin(obj)
```

Description:

Take inverse sine, result in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = asin(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asind** ↑

```
obj = asind(obj)
```

Description:

asind(obj) is the inverse sine, expressed in degrees

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asind(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **asinh** ↑

```
obj = asinh(obj)
```

Description:

asinh(obj) is the inverse hyperbolic sine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = asinh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atan** ↑

```
obj = atan(obj)
```

Description:

Take the tangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = atan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atand** ↑

```
obj = atand(obj)
```

Description:

atand(obj) is the inverse tangent, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = atand(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **atanh** ↑

```
obj = atanh(obj)
```

Description:

atanh(obj) is the inverse hyperbolic tangent

Input:

```
- obj : An object of class nb_objectInExpr
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Examples:

```
out = atanh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **avgOAF** ↑

```
obj = avgOAF(obj,avgFreq)
```

Description:

Take the average over a lower frequency.

Input:

```
- obj : An object of class nb_objectInExpr  
- avgFreq : Any
```

Output:

```
- obj : An object of class nb_objectInExpr
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **bkfilter** ↑

```
obj = bkfilter(obj,low,high)
```

Description:

Do band pass filtering

Input:

- obj : An object of class nb_objectInExpr
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **bkfilterls** ↑

```
obj = bkfilterls(obj,low,high)
```

Description:

Do one sided band pass-filtering.

Input:

- obj : An object of class nb_objectInExpr
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **callfun** ↑

```
obj      = callfun(obj,'func',func)
obj      = callfun(obj,another,'func',func)
obj      = callfun(obj,varargin,'func',func)
varargout = callfun(obj,varargin,'func',func)
```

Description:

Call a in-built or user defined function on the object.

Input:

- obj : An object of class nb_logicalInExpr.

Optional input:

- 'func' : Either a function handle or a one line char with the name of a function (may be user defined in both cases). If not provided the function $\theta(x)x$ will be used.
- 'frequency' : Any
- 'interpolateDate' : Any
- varargin : Optional inputs given as extra inputs to the function func. May be of any class.

Output:

- varargout : The output will be given as return by the func function when applied to the input objects.

Examples:

```
d1 = callfun(d,'func',@sin); % Same as d1 = sin(d) !
d2 = callfun(d,d,'func','plus') % Same as d2 = d + d !
```

Written by Kenneth Sæterhagen Paulsen

► ceil ↑

```
obj = ceil(obj)
```

Description:

ceil(obj) rounds the elements of obj to the nearest integers towards infinity.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = ceil(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **conj** ↑

```
obj = conj(obj)
```

Description:

conj(obj) is the complex conjugate

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = conj(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **convert** ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_logicalInExpr object

Input:

- obj : An object of class nb_logicalInExpr
- freq : Any
- method : Any

Optional input:

- Any

Output:

- obj : An nb_logicalInExpr object.

Written by Kenneth S. Paulsen

► **corr ↑**

obj = corr(obj)

Description:

Correlation

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cos ↑**

obj = cos(obj)

Description:

Cosine

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cos(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_OBJECTINEXPR
```

► **cosd** ↑

```
obj = cosd(obj)
```

Description:

`cosd(obj)` is the cosine of the elements of `obj`, expressed in degrees.

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = cosd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cosh** ↑

```
obj = cosh(obj)
```

Description:

`cosh(obj)` is the hyperbolic cosine.

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = cosh(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR
```

► **cot** ↑

```
obj = cot(obj)
```

Description:

Cotangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cot(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cotd** ↑

```
obj = cotd(obj)
```

Description:

cotd(obj) is the cotangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = cotd(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR
```

► **coth** ↑

```
obj = coth(obj)
```

Description:

coth(obj) is the hyperbolic cotangent of the elements of obj.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = coth(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cov** ↑

```
obj = cov(obj)
```

Description:

Covariance

Input:

obj : An object of class nb_objectInExpr

Output:

d : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **csc** ↑

```
obj = csc(obj)
```

Description:

Cosecant of argument in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = csc(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **cscd** ↑

```
obj = cscd(obj)
```

Description:

cscd(obj) is the cosecant, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = cscd(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **csch** ↑

```
obj = csch(obj)
```

Description:

`csch(obj)` is the hyperbolic cosecant of the elements of `obj`

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = csch(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **cumprod** ↑

```
obj = cumprod(obj, dim)
```

Description:

Cumulativ product

Input:

- `obj` : An object of class `nb_objectInExpr`
- `dim` : In which dimension the cumulativ product should be calculated. Default is the first dimension.

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
obj = cumprod(obj, 1);
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **cumsum** ↑

```
obj = cumsum(obj,dim)
```

Description:

Cumulative sum

Input:

- obj : An object of class nb_objectInExpr
- dim : In which dimension the cumulative sum should be calculated. Default is the first dimension.

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = cumsum(obj,1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **demean** ↑

```
obj = demean(data,~)
```

Description:

- Demeans along the dimension you choose.

Input:

- obj : A nb_objectInExpr object
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- obj : A nb_objectInExpr object.

See also:

[sgrowth](#), [subAvg](#).

Written by Tobias Ingebrigtsen

Inherited from superclass NB_OBJECTINEXPR

► **demeanMoving** ↑

```
obj = demeanMoving(obj,backward)
obj = demeanMoving(obj,backward,forward)
```

Description:

- Demeans data with a moving average process.

Input:

- obj : A nb_objectInExpr object
- backward : Number of periods backward in time to calculate the moving average
- forward : Number of periods forward in time to calculate the moving average

Output:

- obj : A nb_objectInExpr object.

See also:

[nb_logicalInExpr.demean](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **denton** ↑

```
obj = denton(obj,z,k,type,d)
```

Description:

The Denton method of transforming a series from low to high frequency.

See Denton (1971), Adjustment of Monthly or Quarterly Series to Annual Totals: An Approach Based on Quadratic Minimization.

Input:

- obj : A nb_logicalInExpr object.
- z : Any
- k : Any
- type : Any
- d : Any

Output:

- x : A nb_logicalInExpr object.

See also:

[nb_denton](#)

Written by Kenneth Sæterhagen Paulsen

► **deptcat** ↑

obj = deptcat(a,b,varargin)

Description:

Depth concatenation (add pages of different nb_objectInExpr objects)
(No short hand notation)

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional number of nb_objectInExpr objects

Output:

obj : An object of class nb_objectInExpr.

Examples:

```
obj = deptcat(a,b);  
obj = deptcat(a,b,c);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **detrend** ↑

```
obj = detrend(obj,method,output,varargin)
```

Description:

De-trend.

Input:

- obj : A nb_objectInExpr object.
- method : Any
- output : Any

Optional input:

Any

Output:

- obj : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **diff ↑**

```
obj = diff(obj)
obj = diff(obj,lag)
obj = diff(obj,lag,skipNaN)
```

Description:

Diff method.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- skipNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = diff(obj);
obj = diff(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **disp** ↑

```
disp(obj)
```

Description:

Display object (On the commandline)

Input:

obj : An object of class nb_logicalInExpr

Output:

The object display on the command line. (When not ending the command with an semicolon)

Examples:

```
obj
```

Written by Kenneth S. Paulsen

► **egrowth** ↑

```
obj = egrowth(obj,lag,stripNaN)
```

Description:

Calculate exact growth, using the formula: $(x(t) - x(t-1))/x(t-1)$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = egrowth(obj);
obj = egrowth(obj, 4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **epcn** ↑

```
obj = epcn(obj, lag, stripNaN)
```

Description:

Calculate exact percentage growth, using the formula:
 $100 * (x(t) - x(t-1)) / x(t-1)$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = epcn(obj); % period-on-period growth
obj = epcn(obj, 4); % 4-periods growth
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **eq** ↑

```
a = eq(a, b)
```

Description:

This method is intended to test if two nb_math_ts objects are equal. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar.
- b : An object of class nb_objectInExpr or a scalar.

Output:

- a : An object of class nb_objectInExpr

Examples:

```
obj = a == b;  
obj = 2 == b;  
obj = a == 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **exp ↑**

```
obj = exp(obj)
```

Description:

Exponential function.

Input:

- obj : An object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr.

Examples:

```
obj = exp(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **expand ↑**

```
obj = expand(obj,newStartDate,newEndDate,type,warningOff)
```

Description:

This method expands a nb_math_ts object, which means we need to take max of existing end date and the new end date.

Input:

- obj : An object of class nb_logicalInExpr
- newStartDate : Any
- newEndDate : Any
- type : Any
- warningOff : Any

Output:

- obj : An nb_logicalInExpr object.

Written by Kenneth S. Paulsen

► **expml** ↑

obj = expml(obj)

Description:

expml(obj) computes $\exp(\text{obj}) - 1$, compensating for the roundoff in $\exp(\text{obj})$.

(For small real X, expml(X) should be approximately X, whereas the computed value of EXP(X)-1 can be zero or have high relative error.)

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = expml(obj);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **extrapolate** ↑

```
obj = extrapolate(obj,toDate)
obj = extrapolate(obj,toDate,varargin)
```

Description:

Extrapolate the time-series.

Input:

- obj : An object of class nb_logicalInExpr
- toDate : Any

Optional inputs:

- Any

Output:

- obj : An nb_logicalInExpr object.

Written by Kenneth S. Paulsen

► **fillNaN ↑**

```
obj = fillNaN(obj)
obj = fillNaN(obj,date)
```

Description:

Fill in for nan values with last valid observation.

Input:

- obj : An object of class nb_logicalInExpr.
- date : Any

Output:

- obj : An object of class nb_logicalInExpr.

Examples:

```
d2 = fillNaN(d);
```

Written by Kenneth Sæterhagen Paulsen

► **floor** ↑

```
obj = floor(obj)
```

Description:

`floor(obj)` rounds the data elements of `obj` to the nearest integers towards minus infinity

Input:

- `obj` : An object of class `nb_objectInExpr`

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
out = floor(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **ge** ↑

```
a = ge(a,b)
```

Description:

This method is intended to test if a `nb_math_ts` object is greater than or equal to another. We therefore just return the `a` object here as the date the objects represent does not change in this case.

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar

- `b` : An object of class `nb_objectInExpr` or a scalar

Output:

- `a` : An `nb_objectInExpr` object.

Examples:

```
obj = a >= b;  
obj = 2 >= b;  
obj = a >= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► growth ↑

```
obj = growth(obj)  
obj = growth(obj,lag,stripNaN)
```

Description:

Calculate approx growth, using the formula: $\log(x(t)) - \log(x(t-1))$.

Input:

- obj : An object of class nb_objectInExpr
- lag : Any
- stripNaN : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = growth(obj);  
obj = growth(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► gt ↑

```
a = gt(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is greater than another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a > b;  
obj = 2 > b;  
obj = a > 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **h21** ↑

```
obj = h21(obj,freq,type)
```

Description:

Map high 2 low frequency observation with converting the frequency of the object.

Input:

- obj : An object of class nb_objectInExpr
- freq : Any
- type : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **horzcat** ↑

```
obj = horzcat(a,b,varargin)
```

Description:

Horizontal concatenation ([a,b])

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional number of nb_objectInExpr objects

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = [a,b];  
obj = [a,b,c];
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **horzcatfast** ↑

```
obj = horzcatfast(varargin)
```

Description:

Horizontal concatenation

Input:

- a : An object of class nb_objectInExpr
- b : An object of class nb_objectInExpr
- varargin : Optional number of nb_objectInExpr objects

Output:

- obj : An object of class nb_objectInExpr

See also
nb_logicalInExpr.horzcat

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **hpfilter** ↑

```
obj = hpfilter(obj,lambda)
```

Description:

HP-filtering.

Input:

- obj : An object of class nb_objectInExpr
- lambda : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
gap = hpfilter(data,3000);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **hpfilterls** ↑

```
obj = hpfilterls(obj,lambda)
```

Description:

Do one-sided hp-filtering.

Input:

- obj : An object of class nb_objectInExpr
- lambda : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
gap = hpfilterls(data,3000)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iegrowth** ↑

```
obj = iegrowth(obj,initialValues,periods)
```

Description:

Inverse of exact growth, i.e. the inverse method of the egrowth method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = iegrowth(obj);  
obj = iegrowth(obj,100);  
obj = iegrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iegrowthnan** ↑

```
obj = iegrowthnan(obj)  
obj = iegrowthnan(obj,initialValues,periods)
```

Description:

Inverse of exact growth, i.e. the inverse method of the egrowth method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iepcn** ↑

```
obj = iepcn(obj,initialValues,periods)
```

Description:

Inverse of exact percentage growth, i.e. the inverse method of the epcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = iepcn(obj);  
obj = iepcn(obj,100);  
obj = iepcn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **iepcnnan** ↑

```
obj = iepcnnan(obj)  
obj = iepcnnan(obj,initialValues,periods)
```

Description:

Inverse of exact percentage growth, i.e. the inverse method of the epcn method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► igrowth ↑

```
obj = igrowth(obj,initialValues,periods)
```

Description:

Inverse of log approx. growth, i.e. the inverse method of the growth method

Input:

- obj : An object of class nb_objectInExpr
- InitialValues : Any
- periods : Any

Output:

- obj : An nb_objectInExpr object.

Examples:

```
obj = igrowth(obj);  
obj = igrowth(obj,100);  
obj = igrowth(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► igrowthnan ↑

```
obj = igrowthnan(obj)  
obj = igrowthnan(obj,initialValues,periods)
```

Description:

Inverse of log approx. growth, i.e. the inverse method of the growth method

Input:

```
- obj : An object of class nb_objectInExpr  
- InitialValues : Any  
- periods : Any
```

Output:

```
- obj : An nb_objectInExpr object.
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **interpolate** ↑

```
obj = interpolate(obj,method)
```

Description:

Interpolate.

Input:

```
- data : A nb_objectInExpr object.  
- method : Any
```

Output:

```
- data : A nb_objectInExpr object.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ipcn** ↑

```
obj = ipcn(obj)  
obj = ipcn(obj,initialValues,periods,startAtNaN)
```

Description:

Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method

Input:

```
- obj : An object of class nb_objectInExpr  
- InitialValues : Any  
- periods : Any  
- startAtNaN : Any
```

Output:

```
- obj : An nb_objectInExpr object.
```

Examples:

```
obj = ipcnn(obj);  
obj = ipcnn(obj,100);  
obj = ipcnn(obj,[78,89]);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ipcnnan** ↑

```
obj = ipcnnan(obj)  
obj = ipcnnan(obj,initialValues,periods)
```

Description:

Inverse of percentage log approx. growth, i.e. the inverse method of the pcn method

Input:

```
- obj : An object of class nb_objectInExpr  
- InitialValues : Any  
- periods : Any
```

Output:

```
- obj : An nb_objectInExpr object.
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **isfinite** ↑

```
obj = isfinite(obj)
```

Description:

Is finite.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = isfinite(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **isnan** ↑

```
obj = isnan(obj)
```

Description:

Is nan.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = isnan(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **kurtosis** ↑

```
obj = kurtosis(obj,flag,dim,outputType)
```

Description:

Calculate the kurtosis.

Input:

- obj : An object of class nb_objectInExpr
- flag : Any
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **lag** ↑

obj = lag(obj,periods)

Description:

Lag method.

Input:

- obj : An object of class nb_objectInExpr
- periods : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = lag(obj,1);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ldivide** ↑

```
obj = ldivide(a,b)
```

Description:

Left element-wise division (.\ \backslash). Same as right element-wise (./) division. See rdivide for more.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a.\b;  
obj = 2.\b;  
obj = a.\2;
```

See also:

[rdivide](#), [mldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **le ↑**

```
a = le(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is less than or equal to another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a <= b;  
obj = 2 <= b;  
obj = a <= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **lead** ↑

```
obj = lead(obj,periods)
```

Description:

Lead method.

Input:

- obj : An object of class nb_objectInExpr
- periods : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = lead(obj,1);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log** ↑

```
obj = log(obj)
```

Description:

Log operator

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log(obj);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log10** ↑

```
obj = log10(obj)
```

Description:

Take the common (base 10) log

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **log1p** ↑

```
obj = log1p(obj)
```

Description:

log1p(obj) computes $\text{LOG}(1+xi)$, where xi are the elements of obj.
Complex results are produced if $xi < -1$.

For small real xi, log1p(xi) should be approximately xi, whereas the computed value of $\text{LOG}(1+xi)$ can be zero or have high relative error.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = log1p(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► log2 ↑

```
obj = log2(obj)
```

Description:

log2(obj) is the base 2 logarithm of the elements of obj.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = log2(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► lt ↑

```
a = lt(a,b)
```

Description:

This method is intended to test if a nb_math_ts object is less than another. We therefore just return the a object here as the date the objects represent does not change in this case.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- a : An nb_objectInExpr object.

Examples:

```
obj = a < b;  
obj = 2 < b;  
obj = a < 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mavg** ↑

```
obj = mavg(obj,backward,forward,flag)
```

Description:

Taking moving avarage

Input:

- obj : An object of class nb_objectInExpr
- backward : Any
- forward : Any
- flag : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **max** ↑

```
obj = max(obj,dim,outputType)
```

Description:

Max

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType : - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mean** ↑

obj = mean(obj,dim,outputType)

Description:

Mean

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType : - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **median** ↑

```
obj = median(obj,dim,outputType)
```

Description:

Median.

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **merge2Series** ↑

```
obj = merge2Series(obj,other,method,varargin)
```

Description:

Merge 2 series.

Input:

- obj : An object of class nb_objectInExpr.
- other : An object of class nb_objectInExpr.
- method : Any

Optional input:

- Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **min** ↑

```
obj = min(obj, dim, outputType)
```

Description:

Min

Input:

- obj : An object of class nb_objectInExpr
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **minus** ↑

```
obj = minus(a, b)
```

Description:

Binary subtraction (-).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a-b;
obj = 2-b;
obj = a-2;
```

See also:

[plus](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **mldivide** ↑

```
obj = mldivide(a,b)
```

Description:

Matrix left division (\backslash).

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar
- `b` : An object of class `nb_objectInExpr` or a scalar

Output:

- `obj` : An object of class `nb_objectInExpr`

Examples:

```
obj = 2\obj;
obj = obj\2;
```

See also:

[rdivide](#), [ldivide](#), [mrdivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **mpower** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^)

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = 2^b;  
obj = a^2;
```

See also:

[power](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mrddivide** ↑

obj = mrddivide(a,b)

Description:

Matrix left division (/).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = 2/obj;  
obj = obj/2;
```

See also:

[rdivide](#), [ldivide](#), [mlddivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mstd** ↑

obj = mstd(obj,backward,forward)

Description:

Moving standard deviation

Input:

- obj : An object of class nb_objectInExpr
- backward : Number of periods backward in time to calculate the moving std
- forward : Number of periods forward in time to calculate the moving std

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **mtimes** ↑

obj = power(a,b)

Description:

Matrix multiplication (*).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = 2*b;  
obj = a*2;
```

See also:

[times](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **ne** ↑

```
a = ne(a,b)
```

Description:

This method is intended to test if two `nb_math_ts` objects are not equal. We therefore just return the `a` object here as the date the objects represent does not change in this case.

Input:

- `a` : An object of class `nb_objectInExpr` or a scalar.
- `b` : An object of class `nb_objectInExpr` or a scalar.

Output:

- `a` : An object of class `nb_objectInExpr`

Examples:

```
obj = a ~= b;  
obj = 2 ~= b;  
obj = a ~= 2;
```

Written by Kenneth S. Paulsen

Inherited from superclass `NB_OBJECTINEXPR`

► **not** ↑

```
obj = not(obj)
```

Description:

The `not` operator (`~`)

Input:

- a : An object of class nb_objectInExpr

Output:

- a : An object of class nb_objectInExpr

Examples:

obj = ~obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► or ↑

a = or(a,b)

Description

The or operator (|)

Input:

- a : An object of class nb_objectInExpr

- b : An object of class nb_objectInExpr

Output:

- a : An object of class nb_objectInExpr

Examples:

a = a | b;

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► pca ↑

F = pca(obj)
[F,LAMBDA,R,varF,expl,c,sigma,e,Z] = pca(obj,r,method,varargin)

Description:

Principal Component Analysis.

Input:

- obj : An object of class nb_objectInExpr.
- The rest not important

Output:

- F : A nb_objectInExpr object.
- LAMBDA : A empty nb_objectInExpr object.
- R : A empty nb_objectInExpr object.
- varF : A empty nb_objectInExpr object.
- expl : A empty nb_objectInExpr object.
- c : A empty nb_objectInExpr object.
- sigma : A empty nb_objectInExpr object.
- e : A nb_objectInExpr object.
- Z : A nb_objectInExpr object.

See also:

[nb_pca](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pca_func** ↑

factors = pca_func(varargin)

Description:

Principal Component of selected number of series represented by a set of nb_objectInExpr.

Optional input:

- varargin : Optional number of nb_objectInExpr objects.
- 'numFactors' : Determine number of Factors to report. Standard is first factor only.
- 'unbalanced' : true or false.
- 'whichFactor' : Select the factor to return.

Output:

- factors : An object of class nb_objectInExpr.

See also:

[nb_logicalInExpr.pca](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pcn** ↑

obj = pcn(obj,lag,stripNaN)

Description:

Calculate log approximated percentage growth. Using the formula
 $(\log(t+lag) - \log(t)) * 100$

Input:

- obj : An object of class nb_objectInExpr
- lag : The number of lags. Default is 1.
- stripNaN : Strip nan before calculating the growth rates.

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = pcn(obj); % period-on-period log approx. growth
obj = pcn(obj,4); % 4-periods log approx. growth
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **plus** ↑

obj = plus(a,b)

Description:

Binary addition (+).

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a+b;  
obj = 2+b;  
obj = a+2;
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **pow2** ↑

```
obj = pow2(obj)
```

Description:

pow2(obj) raises the number 2 to the object

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = pow2(obj);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **power** ↑

```
obj = power(a,b)
```

Description:

Element-wise power (.^)

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a.^b;
obj = 2.^b;
obj = a.^2;
```

See also:

[mpower](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **q2y** ↑

obj = q2y(obj)

Description:

Cumulative sum over the last 4 periods.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = q2y(obj)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **rdivide** ↑

```
obj = rdivide(a,b)
```

Description:

Right element-wise division (./)

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = a./b;
obj = 2./b;
obj = a./2;
```

See also:

[mrdivide](#), [mldivide](#), [ldivide](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **reIndex** ↑

```
obj = reIndex(obj,date)
```

Description:

Reindex the data of the object to a new date.

Input:

- obj : An object of class nb_objectInExpr
- date : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **real** ↑

obj = real(obj)

Description:

real(obj) returns the real part

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **reallog** ↑

obj = reallog(obj)

Description:

The real natural logarithm

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = reallog(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **realpow** ↑

```
obj = realpow(a,b)
```

Description:

Real power.

Input:

- a : An object of class nb_objectInExpr or a scalar
- b : An object of class nb_objectInExpr or a scalar

Output:

- obj : An object of class nb_objectInExpr

See also:

[mpower](#), [power](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **realsqrt** ↑

```
obj = realsqrt(obj)
```

Description:

realsqrt(obj) is the the real square root

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = realsqrt(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **ret** ↑

```
obj = ret(obj)
obj = ret(obj,nlag)
obj = ret(obj,nlag,skipNaN)
```

Description:

Calculate return, using the formula: $x(t)/x(t-lag)$.

Input:

- obj : An object of class nb_objectInExpr
- nlag : Any
- skipNaN : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = ret(obj);
obj = ret(obj,4);
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **rlag** ↑

```
obj = rlag(obj,lags)
obj = rlag(obj,lags,periods)
```

Description:

Roll a pattern in the data forward with a given lag.

Input:

- obj : An object of class nb_objectInExpr
- lags : Any
- periods : The extrapolated periods. The end date of the object will be the current end date plus these number of periods. Be aware that trailing nan values in the data will be filled in for even if periods == 0, which is the default.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► round ↑

```
obj = round(obj,value,startDate,endDate,pages)
```

Description:

Round to closes value. I.e. if value is 0.25 it will round to the closes 0.25. E.g. 0.67 will be rounded to 0.75.

Input:

- obj : A nb_objectInExpr object
- value : Any
- startDate : Any
- endDate : Any
- variables : Any

Output:

- obj : A nb_objectInExpr object

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► sec ↑

```
obj = sec(obj)
```

Description:

Secant of argument in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr.

Examples:

obj = sec(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **secd** ↑

obj = secd(obj)

Description:

Secant of argument in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

out = secd(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sech** ↑

obj = sech(obj)

Description:

Hyperbolic secant.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sech(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **set2Value** ↑

```
obj = set2Value(obj,func,value)
```

Description:

Set all values applying to the restriction given by the input func to the provided value.

Input:

- obj : An object of class nb_objectInExpr.
- func : Any
- value : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **setNan2Zero** ↑

```
obj = setNan2Zero(obj)
```

Description:

Set all nan values of the data of the object to 0.

Input:

- obj : An object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sgrowth** ↑

obj = sgrowth(obj,horizon)

Description:

Calculates smoothed 12 or 6 months growth.

$x(*) = ((x13+x14+x15)/(x1+x2+x3)-1)*100$

$x(*) = ((x7+x8+x9)/(x1+x2+x3)-1)*100$

Input:

- obj : A nb_objectInExpr object

- horizon : Any

Output:

- out : A nb_objectInExpr object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sin** ↑

obj = sin(obj)

Description:

Sine of argument in radians.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr.

Examples:

```
obj = sin(obj);  
  
Written by Kenneth S. Paulsen  
  
Inherited from superclass NB_OBJECTINEXPR
```

► **sind** ↑

```
obj = sind(obj)
```

Description:

Sine of argument in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sind(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sinh** ↑

```
obj = sinh(obj)
```

Description:

Hyperbolic sine.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sinh(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR
```

► **skewness** ↑

```
obj = skewness(obj,flag,dim,outputType)
```

Description:

Calculate the skewness

Input:

- obj : An object of class nb_objectInExpr
- flag : Any
- dim : Any
- outputType :
 - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

Square root.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = sqrt(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **std** ↑

```
obj = std(obj,flag,dim,outputType)
```

Description:

Standard deviation

Input:

- obj : An object of class nb_objectInExpr
- flag : Any
- dim : Any
- outputType : - 'nb_math_ts': The result will be an object of class nb_objectInExpr.
 - 'double': The result will be an empty object of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **stdise** ↑

```
obj = stdise(obj,flag)
```

Description:

Standardise

Input :

- obj : An object of class nb_objectInExpr
- flag : Any

Output:

- obj : An nb_objectInExpr object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **subAvg** ↑

obj = subAvg(obj, k, w)

Description:

- Will for the last k periods (including the present) calculate the cumulative sum and then divide by the amount of periods, which gives you the average over those periods.

Input:

- obj : An object of class nb_objectInExpr.
- k : Any
- w : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **subSum** ↑

obj = subSum(obj, k, w)

Description:

Calculates the cumulative sum over the last k periods (including the present period).

Input:

- obj : An object of class nb_objectInExpr.
- k : Any
- w : Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sum** ↑

obj = sum(obj,dim)

Description:

Takes the sum of the object data in the wanted dimension

Input:

- obj : An object of class nb_objectInExpr
- dim : Any

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
obj = sum(obj,1)
obj = sum(obj,2)
obj = sum(obj,3)
```

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **sumOAF** ↑

obj = sumOAF(obj,sumFreq)

Description:

Take the sum over a lower frequency.

Input:

- obj : An object of class nb_objectInExpr
- sumFreq : Any

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **tan ↑**

obj = tan(obj)

Description:

Tangent

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr.

Examples:

obj = tan(obj);

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **tand ↑**

obj = tand(obj)

Description:

tand(obj) is the tangent of the elements of obj, expressed in degrees.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = tand(in);

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR
```

► **tanh** ↑

```
obj = tanh(obj)
```

Description:

Hyperbolic tangent.

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

```
out = tanh(in);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **times** ↑

```
obj = times(a,b)
```

Description:

Element-wise multiplication (.*)

Input:

- a : An object of class nb_objectInExpr or a scalar double
- b : An object of class nb_objectInExpr or a scalar double

Output:

- obj : An object of class nb_objectInExpr

See also:

mtimes

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = -obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **undiff** ↑

obj = undiff(obj,initialValues,periods)

Description:

Inverse of the diff method.

Input:

- obj : An object of class nb_objectInExpr.
- InitialValues : Any.
- periods : Any.

Output:

- Output : An object of class nb_objectInExpr.

See also
nb_logicalInExpr.diff

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **uplus** ↑

obj = uplus(obj)

Description:

Unary plus

Input:

- obj : An object of class nb_objectInExpr

Output:

- obj : An object of class nb_objectInExpr

Examples:

obj = +obj;

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **var** ↑

obj = var(obj, dim, outputType)

Description:

Variance

Input:

- obj : An object of class nb_objectInExpr

- dim : Any

- outputType : - 'nb_math_ts': The result will be an object
of class nb_objectInExpr.

- 'double': The result will be an object
of class nb_objectInExpr.

Output:

- obj : An object of class nb_objectInExpr

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► vertcat ↑

```
obj = vertcat(a,b,varargin)
```

Description:

Vertical concatenation ([a;b])

Input:

- a : An object of class nb_objectInExpr

- b : An object of class nb_objectInExpr

- varargin : Optional numbers of objects of class nb_objectInExpr

Output:

- a : An nb_objectInExpr object

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► window ↑

```
obj = window(obj,startDateWin,endDateWin,Pages)
```

Description:

Narrow down window

Input:

- obj : An object of class nb_logicalInExpr

- startDateWin : Any

- endDateWin : Any

- pages : Any

Output:

- obj : An object of class nb_logicalInExpr

Written by Kenneth S. Paulsen

► **x12** ↑

obj = x12(obj,range,varargin)

Description:

Seasonally adjust

Input:

- obj : An object of class nb_objectInExpr.

- range : Any

Optional input (..., 'propertyName', PropertyValue, ...):

Any

Output:

- obj : An object of class nb_objectInExpr.

Written by Kenneth S. Paulsen

Inherited from superclass NB_OBJECTINEXPR

► **x12Census** ↑

obj = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj)
[obj,x,outputfile,errorfile,model] = x12Census(obj,varargin)

Description:

Do x12 Census

Input:

- obj : An object of class nb_objectInExpr.

Optional input (..., 'propertyName', PropertyValue, ...):

Any.

Output:

- obj : An object of class nb_objectInExpr.
- x : An object of class nb_objectInExpr.
- outputfile : ''
- errorfile : {}
- mdl : []

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_OBJECTINEXPR

25.9 Double functions

- `bkfilter`
- `createDependentDummy`
- `epcn`
- `growth`
- `hpfilter1s`
- `iegrowthnan`
- `iepcnnan`
- `igrowthnan`
- `ipcnnan`
- `lead`
- `nb_arRebalance`
- `nb_bkfilter1s`
- `nb_calculateVariation`
- `nb_demean`
- `nb_diff`
- `nb_expandCov`
- `nb_exponentialDecayingMean`
- `nb_findBasis`
- `nb_hpfilter`
- `nb_interpolate`
- `nb_isScalarNumber`
- `nb_isScalarPositiveNumber`
- `nb_iswholenumber`
- `nb_lagmatrix`
- `nb_linearFilter1s`
- `nb_mavg`
- `nb_mlag`
- `bkfilter1s`
- `egrowth`
- `exptrend`
- `hpfilter`
- `iegrowth`
- `iepcn`
- `igrowth`
- `ipcn`
- `lag`
- `nb_addLeads`
- `nb_bkfilter`
- `nb_breakAdj`
- `nb_closestTo`
- `nb_denton`
- `nb_double2cell`
- `nb_exponentialDecay`
- `nb_exponentialDecayingSum`
- `nb_histcounts`
- `nb_index`
- `nb_isScalarInteger`
- `nb_isScalarNumberClosed`
- `nb_iswholecolumn`
- `nb_iswholerow`
- `nb_linearFilter`
- `nb_linespace`
- `nb_meanHist`
- `nb_mlead`

- nb_msum
 - nb_num2cell
 - nb_percentiles
 - nb_rlag
 - nb_seasonalDummy
 - nb_slag
 - nb_splitSample
 - nb_subSum
 - nb_undiff
 - nb_x12.x12
 - q2y
 - steady_state
 - steady_state_first
 - nb_nanmavg
 - nb_num2str
 - nb_reduceCov
 - nb_rref
 - nb_sgrowth
 - nb_spline
 - nb_subAvg
 - nb_trimr
 - nb_undifnan
 - pcn
 - steady_state
-

► **nb_x12.x12 ↑**

```
[yData,outData,outp,err,mdl] = x12(data,startdate,dummy,options)
```

Description:

Do x12 Census adjustment to time series using x12awin.exe.

This is a file copied from the IRIS Toolbox an adapted to the NB Toolbox.

x12awin.exe is a program developed by:

U. S. Department of Commerce, U. S. Census Bureau

X-12-ARIMA quarterly seasonal adjustment Method,
Release Version 0.3 Build 192

This method modifies the X-11 variant of Census Method II
by J. Shiskin A.H. Young and J.C. Musgrave of February, 1967.
and the X-11-ARIMA program based on the methodological research
developed by Estela Bee Dagum, Chief of the Seasonal Adjustment
and Time Series Staff of Statistics Canada, September, 1979.

This version of X-12-ARIMA includes an automatic ARIMA model
selection procedure based largely on the procedure of Gomez and
Maravall (1998) as implemented in TRAMO (1996).

Input:

- data : double
- startdate : An nb_date object.

```
- dummy      : double.  
- options    : A structure with options.
```

Output:

```
- yData      : A double with the original data appended backcast  
and forecast.  
- outData    : The adjusted data.  
- outp       : A cellstr array with the output files from  
x12awin.exe  
- err        : A cellstr array with error files from x12awin.exe  
- mdl        : A struct with ARIMA model estimates.
```

Written by Jaromir Benes
Modified by Kenneth SÃ¶terhagen Paulsen

◆ **bkfilter**

```
x = bkfilter(y,low,high)
```

See also:

[nb_bkfilter](#)

Written by Kenneth SÃ¶terhagen Paulsen

◆ **bkfilterls**

```
x = bkfilterls(y,low,high)
```

See also:

[nb_bkfilterls](#)

Written by Kenneth SÃ¶terhagen Paulsen

◆ **createDependentDummy**

```
y = createVarDummy(y,condition)  
y = createVarDummy(y,condition,scalar)
```

Description:

Create dummy series based on time-series stored as double.

Input:

- obj : An double with size nObs x nVars x nPages.
- condition : Name of the added dummy variable.
 - '<' : Less than.
 - '>' : Greater than.
 - '<=' : Less than or equal to.
 - '>=' : Less than or equal to.
 - '==' : Equal to.
 - '~=': Not equal to.
- scalar : A scalar double. Default is 0.

Output:

- obj : An double with size nObs x nVars x nPages.

Examples:

```
y = [1,2;-3,1;-1,3];
y = createDependentDummy(y,'>',0);
```

See also:

[nb_math_ts.createDependentDummy](#)

Written by Kenneth Sæterhagen Paulsen

◆ egrowth

```
dout = egrowth(din,t)
```

Description:

This function uses the standard growth formula as opposed to the log growth economist typically use.

See also:

[growth, iegrowth](#)

Written by Kenneth Sæterhagen Paulsen

◆ epcn

```
dout = epcn(din,t)
```

Description:

This function uses the standard growth formula as opposed to the log growth economist typically use.

See also:

[iepcn](#), [egrowth](#), [iegrowth](#)

Written by Kenneth Sætherhagen Paulsen

◆ **exptrend**

```
trend = exptrend(series)
trend = exptrend(series,weight,numInitialisationAvg)
```

Description:

Exponential smoothing.

Translated from Gauss code originally produced by Anne-Sofie Jore.

Input:

- series : The series to be smoothed (each column is smoothed using the parameter weight). A double with size T x N x P.
- weight : The amount of weight attached to the immediately lagged value or (1-weight) attached to last period's trend value. Default is 0.5.
- numInitialisationAvg : The number of observations to average across at the beginning of the sample

Output:

- trend : The trend. A double with size T x N x P.

See also:

[hpfilter](#)

Written by Christie Smith
Edited by Kenneth Sætherhagen Paulsen

◆ **growth**

```
dout = growth(din,t)
```

See also:

[egrowth](#), [igrowth](#)

Written by Kenneth Sæterhagen Paulsen

◆ **hpfilter**

```
y = hpfilter(x,lamb)
y = hpfilterls(x,lamb,perc,fcstLen)
```

Description:

Hodrick-Prescott filter. Handle nan.

See page 3 of Cornea-Madeira (2016), "The Explicit Formula for the Hodrick-Prescott Filter in Finite Sample".

Input:

- x : A timeseries, as a nobs x 1 x npage double.
- lamb : The lambda of the hp-filter
- perc : Set to true to calculate the gap as (gap/trend)*100.
- fcstLen : The forecast length. Extrapolates the original series using the average of the last 4 periods. Default is 0.

Output:

- y : The cyclical component of the x-series

Examples:

```
y = hpfilter(x,400);
```

Written by Kenneth Sæterhagen Paulsen

◆ **hpfilterls**

```
y = hpfilterls(x,lamb)
y = hpfilterls(x,lamb,perc,fcstLen)
```

Description:

One sided Hodrick-Prescott filter. Handle nan.

Input:

```
- x : A timeseries, as a nobs x nvars x npage double.  
- lamb : The lambda of the hp-filter.  
- perc : Set to true to calculate the gap as (gap/trend)*100.  
- fcstLen : The forecast length. Extrapolates the original series  
using the average of the last 4 periods. Default is 0.
```

Output:

```
- y : The cyclical component of the x-series
```

Examples:

```
y = hpfilter(x,400);
```

See also:

[hpfilter](#)

Written by Kenneth Sæterhagen Paulsen

◆ **iegrowth**

```
dout = iegrowth(DX,X,periods)
```

Description:

Construct indicies based on initial values and timeseries which represent the series growth. Inverse of exact growth, i.e. the inverse method of the egrowth method of the double class

Input:

```
- din : A double matrix with dimensions [r,c,p].  
- t : A double matrix with dimensions [r,c,p].  
- periods : The number of periods the din has been growthed over.
```

Output:

```
- dout : A double matrix with dimensions [r,c,p].
```

See also:

[egrowth](#), [igrowth](#)

Written by Kenneth Sæterhagen Paulsen

◆ **iegrowthnan**

- dout = iegrowthnan(din,t,periods)

Description:

- Uses iegrowth, but is robust for trailing and leading NaNs. It also checks whether theres any NaNs in between observations in the double, and returns an error if it is the case.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

Examples:

```
x = rand(3,3,3);  
y = nan(1,3,3);  
t = [y;x;y];  
  
iegrowthnan(t,100)
```

See also:

[egrowth](#), [iegrowth](#), [growth](#)

Written by Tobias Ingebrigtsen

◆ **iepcn**

dout = iepcn(DX,X,periods)

Description:

Construct indicies based on initial values and series which represent the series growth. Inverse of exact percentage growth, i.e. the inverse method of the epcn method of the double class.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

See also:

[egrowth](#), [iegrowth](#)

Written by Kenneth Sætherhagen Paulsen

◆ iepcnnan

- dout = iepcnnan(din,t,periods)

Description:

- Uses iepcn, but is robust for trailing and leading NaNs. It also checks whether theres any NaNs in between observations in the double, and returns an error if it is the case.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

See also:

[epcn](#), [iepcn](#)

Written by Kenneth Sætherhagen Paulsen

◆ igrowth

dout = igrowth(DX,X,periods)

Description:

Construct indicies based on initial values and time-series which represent the series growth. Inverse of log approx. growth, i.e. the inverse method of the growth method of the double class

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

See also:

[growth](#), [iegrowth](#)

Written by Kenneth Sæterhagen Paulsen

◆ **igrowthnan**

- dout = igrowthnan(din,t,periods)

Description:

- Uses igrowth, but is robust for trailing and leading NaNs. It also checks whether theres any NaNs in between observations in the double, and returns an error if it is the case.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

Examples:

```
x = rand(3,3,3);
y = nan(1,3,3);
t = [y;x;y];

igrowthnan(t,100)
```

See also:

[egrowth](#), [igrowth](#), [growth](#)

Written by Tobias Ingebrigtsen

◆ ipcn

```
dout = ipcn(DX,X,periods)
```

Description:

Construct indicies based on initial values and series which represent the series growth. Inverse of exact growth, i.e. the inverse method of the pcn method of the double class.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

See also:

[growth](#), [igrowth](#)

Written by Kenneth Sæterhagen Paulsen

◆ ipcnnan

```
- dout = ipcnnan(din,t,periods)
```

Description:

- Uses ipcn, but is robust for trailing and leading NaNs. It also checks whether theres any NaNs in between observations in the double, and returns an error if it is the case.

Input:

- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.

Output:

- dout : A double matrix with dimensions [r,c,p].

See also:

[pcn](#), [ipcn](#)

Written by Kenneth Sætherhagen Paulsen

◆ lag

```
xout = lag(xin,t)
xout = lag(xin,t,d)
```

Input:

- xin : The series to lag, as a nObs x nVar x nPage double.
- t : The number of periods to lag the series.
- d : The number to fill in for the first t periods. Default is nan.

Output:

- xout : The series lagged.

See also:

[lead](#)

Written by Kenneth Sætherhagen Paulsen

◆ lead

```
xout = lead(xin,t)
```

Input:

- xin : The series to lead, as a nObs x nVar x nPage double.
- t : The number of periods to lead the series.

Output:

- xout : The series leaded.

See also:

[lag](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_addLeads

```
Y = nb_addLeads(X,nLeads,dimExpand,dim)
```

Written by Kenneth Sølnerhagen Paulsen

◆ nb_arRebalance

```
X = nb_arRebalance(X)
```

Description:

Fill in for leading and trailing nan values using a AR(1) with automatic selection of the level of integration of the series.

Input:

- X : A T x 1 double storing the data of the time-series.

Output:

- X : A T x 1 double storing the balanced dataset of the time-series.

See also:

[nb_ts.rebalance](#)

Written by Kenneth Sølnerhagen Paulsen

◆ nb_bkfilter

```
x = nb_bkfilter(y,low,high)
```

Description:

Implements the approximation to the band pass filter found in "The Band Pass Filter" by Lawrence J.Christiano and Terry J. Fitzgerald (1999).

Inputs:

- y : A timeseries, as a nobs x nvars x npage double.
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

```
- x      : double (nobs x 1 x npage) containing filtered data.
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_bkfilter1s

```
x = nb_bkfilter1s(y,low,high)
```

Description:

One sided band-pass filter. For more on the filter see nb_bkfilter.
Handle nan values.

Inputs:

- y : A timeseries, as a nobs x nvars x npage double.
- low : Lowest frequency. > 2
- high : Highest frequency. > low

Output:

- x : Double (nobs x 1 x npage) containing filtered data

See also:

[nb_bkfilter](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_breakAdj

```
obj = breakAdj(obj,method,varargin)
```

Description:

The intent of this function is to set observations around break points to nan, and interpolate the observation at these break-points.

Input:

- d : An double with size T x N x P. Interpolation is done along the first dimension.
- method : The wanted interpolation method to use. See the nb_interpolate function.
- indexes : A vector of integers or a logical array of length T for where to set the rows of d to nan.

Output:

- d : An double with size T x N x P.

See also:

[nb_interpolate](#)

Written by Kenneth S. Paulsen

◆ **nb_calculateVariation**

```
[dataLow, dataHigh, dataMAVG, dataMSTD] =...
    nb_calculateVariation(data)
```

Description:

Get some statistical properties of dataseries stored in an nb_ts object.

If number of arguments out is 1, all the statistical properties are stored in one nb_ts object. See the outputs for more.

Input:

- data : An object of class nb_ts

Output:

- dataLow : An object of class nb_ts with the moving average minus the moving standard deviation of the dataseries
- dataHigh : An object of class nb_ts with the moving average plus the moving standard deviation of the dataseries
- dataMAVG : An object of class nb_ts with the 10-year moving average of the dataseries
- dataMSTD : An object of class nb_ts with the 10-year moving standard deviation of the dataseries

Examples:

```
[low, high] = nb_calculateVariation(data);
```

Written by Kenneth S. Paulsen

◆ **nb_closestTo**

```
m = nb_closestTo(X,x,dim,method)
```

Description:

Find the closes path in the multidimension matrix X to the the multidimension matrix x. x must have only dimension 1 in the dimension dim!

Input:

- X : A matrix with dimension less than 4.
- x : A matrix only dimension 1 in the dimension dim. Otherwise have the same dimensions as X.
- dim : The dimension to pick the closest to. An intger between 1 and 4.
- method : Either 'abs' (absolute deviation) or 'square' (square deviation)

Output:

m : The "element" that was closest to x.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_demean**

```
demean = nb_demean(data,dim)
```

Description:

- Demeans data along the dimension you choose.

Input:

- data : A r*c*p double matrix
- dim : A double corresponding to the dimension you want to take the average over. Default is 1.

Output:

- demean : A r*c*p double matrix.

See also:

[nb_subAvg](#), [nb_subSum](#).

Written by Tobias Ingebrigtsen

◆ nb_denton

```
x = nb_denton(y,z,k,type,d)
```

Description:

The Denton method of transforming a series from low to high frequency.

See Denton (1971), Adjustment of Monthly or Quarterly Series to Annual Totals: An Approach Based on Quadratic Minimization.

Input:

- y : A nobs x nvars x npage double with the main variable to transform.
- z : A nobs*k x nvars x npage double with observations to add judgment.
- k : The number intra low frequency observations. If you have annual data and want out quarterly data use 4.
- type : Either 'sum', 'average', 'first' or 'last'. 'average' is default.
- d : 1 : first differences, 2 : second differences.

Output:

- x : Output series as a nobs*k x nvars x npage double.

See also:

[nb_ts.convert](#), [nb_ts.denton](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_diff

```
dout = nb_diff(din,t)
```

Description:

Difference operator $x(s) - x(s-t)$. dout will append nan at the start to preserve the same size of dout as din.

See also:

[growth](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_double2cell

```
c = nb_double2cell(d,precision)
```

Description:

Convert a double matrix to a cellstr matrix with a given precision.

Input:

- d : A double.
- precision : E.g. '%6.6f' or 4 (number of digits)

Output:

- c : A cell matrix

Written by Kenneth Sæterhagen Paulsen

◆ nb_expandCov

```
C = nb_expandCov(par,n)
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_exponentialDecay

```
out = nb_exponentialDecay(in,lambda)
```

Description:

Exponential decay past observation of a series.

Input:

- in : A nobs x nvar double.
- lambda : A 1 x 1 double between 0 and 1.

Output:

- out : A nobs x nvar double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_exponentialDecayingMean

```
out = nb_exponentialDecayingMean(in, lambda, dim)
```

Description:

Calculate mean using exponential decaying of past observation of a series.

Input:

- in : A nobs x nvar x npage double.
- lambda : A 1 x 1 double between 0 and 1.
- dim : Which dimension to take the mean over. 1, 2 or 3. Default is 1.

Output:

- out : A nobs x nvar x npage double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_exponentialDecayingSum

```
out = nb_exponentialDecayingSum(in, lambda)
```

Description:

Calculate sum using exponential decaying of past observation of a series.

Input:

- in : A nobs x nvar x npage double.
- lambda : A 1 x 1 double between 0 and 1.
- dim : Which dimension to take the mean over. 1, 2 or 3.

Output:

- out : A nobs x nvar x npage double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_findBasis

```
[i,B] = nb_findBasis(A)
```

Description:

Find one basis of the linear system represented by A, and return the location of those equations.

Input:

- A : A NxM double matrix, where N >= M

Output:

- i : The index of the basis of the linear system represented by A.
- B : The basis of the linear system represented by A.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_histcounts**

```
[index,centered] = nb_histcounts(x,interval,centered)
```

Description:

Order the observation (dim1) into bins using the specified interval

Input:

- obj : An object of class double
- interval : A 1 x nBins double
- centered : A 1 x nBins double with the centered bins value.

Output:

- index : Number of counts in each bin. As a 1 x nBins double.
- centered : The centered location of the bins of the elements in index. As a 1 x nBins double.

Examples:

```
obj = nb_data.rand(1,100,4,2);
ind1 = nb_histcounts(obj.data,0.1:0.1:1)
ind2 = nb_histcounts(obj.data,0.1:0.1:1,0.05:0.1:1)
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_hpfILTER**

```
tau = nb_hpfILTER(y,alpha)
```

Description:

Hodrick-Prescott filter. Handle nan.

This is slower than the hpfilter function.

See the "The exact weights of the HP filter" section in Cornea-Madeira (2016), "The Explicit Formula for the Hodrick-Prescott Filter in Finite Sample".

Input:

- x : A time-series, as a nobs x 1 x npage double.
- alpha : The smoothness parameter of the hp-filter.

Output:

- y : The cyclical component of the y-series.

Examples:

```
tau = nb_hpfilter(y, 400);
```

See also:

[hpfilter](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_index

```
d = nb_index(d, ind1, ind2, ind3)
```

Description:**Input:**

- d : A double.
- ind1 : Indices in the first dimension. As a string. E.g. '1:3','1', '1:end' or ':'.
- ind2 : Indices in the second dimension. As a string. E.g. '1:3','1', '1:end' or ':'.
- ind3 : Indices in the third dimension. As a string. E.g. '1:3','1', '1:end' or ':'.

Output:

- d : A double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_interpolate

```
data = nb_interpolate(data)
```

Description:

Interpolate the data given as a double. Will discard leading and trailing nan values.

Input:

- data : A double
- method :
 - > 'nearest' : Nearest neighbor interpolation
 - > 'linear' : Linear interpolation. Default
 - > 'spline' : Piecewise cubic spline interpolation (SPLINE)
 - > 'pchip' : Shape-preserving piecewise cubic interpolation
 - > 'cubic' : Same as 'pchip'
 - > 'v5cubic' : The cubic interpolation from MATLAB 5, which does not extrapolate and uses 'spline' if X is not equally spaced.

Output:

- data : A double

See also:

[interp1](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_isScalarInteger

```
ret = nb_isScalarInteger(x, low, upp)
```

Description:

Test if x is a scalar integer.

Input:

- x : Any type
- low : If x is a number, it will test if $x > \text{low}$ if this input is given.
- upp : If x is a number, it will test if $x < \text{upp}$ if this input is given.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isScalarNumber

```
ret = nb_isScalarNumber(x,low,upp)
```

Description:

Test if x is a scalar number.

Input:

- x : Any type
- low : If x is a number, it will test if $x > \text{low}$ if this input is given.
- upp : If x is a number, it will test if $x < \text{upp}$ if this input is given.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isScalarNumberClosed

```
ret = nb_isScalarNumberClosed(x,low,upp)
```

Description:

Test if x is a scalar number and it is in the closed interval [low,upp].

Input:

- x : Any type
- low : If x is a number, it will test if $x \geq \text{low}$ if this input is given.
- upp : If x is a number, it will test if $x \leq \text{upp}$ if this input is given.

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isScalarPositiveNumber

```
ret = nb_isScalarPositiveNumber(x)
```

Description:

Test if x is a scalar positive number.

Input:

- x : Any type

Output:

- ret : true or false.

Written by Kenneth Sætherhagen Paulsen

◆ nb_iswholecolumn

```
ret = nb_iswholecolumn(input)
```

Description:

True for a column of whole numbers

For example,

nb_iswholecolumn([pi NaN 3 -Inf]') is 0.

nb_iswholecolumn([4 2 3 1]') is 1.

Input:

- in : Any

Output:

- ret : an array that contains 1's where
the elements of X are whole numbers and 0's where they are not.

Written by Henrik Halvorsen Hortemo

◆ nb_iswholenumber

```
ret = nb_iswholenumber(input)
```

Description:

True for whole numbers

For example, nb_iswholenumber([pi NaN 3 -Inf]) is [0 0 1 0].

Input:

- in : Any

Output:

- ret : an array that contains 1's where the elements of X are whole numbers and 0's where they are not.

Written by Henrik Halvorsen Hortemo

◆ nb_iswholerow

```
ret = nb_iswholerow(input)
```

Description:

True for a row of whole numbers

For example,

nb_iswholerow([pi NaN 3 -Inf]) is 0.
nb_iswholerow([4 2 3 1]) is 1.

Input:

- in : Any

Output:

- ret : an array that contains 1's where the elements of X are whole numbers and 0's where they are not.

Written by Henrik Halvorsen Hortemo

◆ nb_lagmatrix

Syntax :

```
[output_matrix] = nb_lagmatrix(input_matrix, lags)
```

Description:

For all values in "lags", this function creates the lags of "input_matrix". The result, "output_matrix" contains all lagged values in the order given in lags.

Inputs:

- input_matrix : A matrix

- lags : lags to be computed. Can be a sequence, e.g 0:2, a matrix, e.g. [0, 2, 3], or an integer value.

Outputs:

- `output_matrix` : A matrix containing all the lags of "input_matrix" that are given in "lags".

Example:

```
With lags = 0:2 and input matrix = [1 2; 3 4; 5 6], nb_lagmatrix returns  
  
output_matrix = 1 2 NaN NaN NaN NaN  
                 3 4 1 2 NaN NaN  
                 5 6 3 4 1 1
```

Written by Maximilian Schräder

◆ **nb_linearFilter**

```
y = nb_linearFilter(x)
```

Description:

Linear filter

Input:

- `x` : A double storing timeseries.

Output:

- `y` : The cyclical component of the x-series

Written by Kenneth Sæterhagen Paulsen

◆ **nb_linearFilter1s**

```
y = nb_linearFilter1s(x)
```

Description:

One sided "linear" filter

Input:

- `x` : A double storing timeseries.

Output:

- `y` : The cyclical component of the x-series

Written by Kenneth Sæterhagen Paulsen

◆ nb_linespace

```
l = nb_linespace(x,y,n)
```

Description:

Generates n points between x and y. x and y must be vectors with same size.

Input:

- x : A m x 1 double or a 1 x m double. Must match y.
- y : A m x 1 double or a 1 x m double. Must match x.
- n : Number of generated points. Default is 100.

Output:

- l : A m x n double or a n x m double dependent on the input.

Examples:**See also:**

Written by Kenneth Sæterhagen Paulsen

◆ nb_mavg

```
xout = nb_mavg(xin,backward,forward,flag)
```

Description:

Smooths dataseries. Moving average

Input:

- xin : The data series to smooth. As a double.
- backward : The number of elements backward to include in moving average.
- forward : The number of elements forward to include in moving average.

Examples:

```
newseries = nb_mavg(x,9,0); % 10-element moving average
```

Written by Kenneth S. Paulsen

◆ nb_meanHist

```
meanVector = nb_meanHist(var,periods)
```

Description:

Calculates the mean and return a M X periods vector of the mean value. Where M is the size of the second dimension of the input

Input:

- var : The data series. As a double
- periods : Size of the first dimension of the output double

Output:

- meanVector : An M X periods vector of the mean value. Where M is the size of the second dimension of the input

Examples:

```
meanVector = nb_meanHist(x,10)
```

Written by Kenneth S. Paulsen

◆ nb_mlag

```
xlag = nb_mlag(x,nlag)
```

Description:

Type = 'lagFast':

Creates a matrix of size(x,1) x size(x,2)*nlag. I.e.
[x1(t-1), x1(t-2), ... x1(t-nlag), x2(t-1), ... x2(t-nlag)]

Type = 'varFast':

Creates a matrix of size(x,1) x size(x,2)*nlag. I.e.
[x1(t-1), ... xp(t-1), ... , x1(t-nlag), ... xp(t-nlag)]

Input:

- x : A double
- nlag : The number of lags. Default is 1. Could be a 1 x nvar double.
- type : Either 'lagFast' (default) or 'varFast'

Output:

- xlag : A double

Examples:

```
xlag = nb_mlag(rand(20,2),4)
```

Written by Kenneth S. Paulsen

◆ nb_mlead

```
xout = nb_mlead(xin,nlead)
```

Description:

Type = 'lagFast':

Creates a matrix of size(x,1) x size(x,2)*nlead. I.e.
[x1(t+1), x1(t+2), ... x1(t+nlead), x2(t+1), ... x2(t+nlead)]

Type = 'varFast':

Creates a matrix of size(x,1) x size(x,2)*nlead. I.e.
[x1(t+1), ... xp(t+1), ... , x1(t+nlead), ... xp(t+nlead)]

Input:

- xin : A double

- nlead : The number of lags. Default is 1. Could be a 1 x nvar double.

- type : Either 'lagFast' (default) or 'varFast'

Output:

- xout : A double

Examples:

```
xout = nb_mlead(rand(20,2),4)
```

Written by Kenneth S. Paulsen

◆ nb_msum

```
xout = nb_msum(xin,backward,forward)
```

Description:

Moving sum

Input:

- `xin` : The data series, as a double.
- `backward` : The number of elements backward to include in moving sum.
- `forward` : The number of elements forward to include in moving sum.

Examples:

```
newseries = nb_msum(x, 9, 0); % 10-element moving sum
```

Written by Kenneth S. Paulsen

◆ **nb_nanmavg**

```
xoutnan = nb_nanmavg(xin,backward,forward,flag)
```

Description:

Smooths dataseries. Moving average. Handles leading and trailing nans.

Input:

- `xin` : The data series to smooth. As a double.
- `backward` : The number of elements backward to include in moving average.
- `forward` : The number of elements forward to include in moving average.

Examples:

```
newseries = nb_nanmavg(x, 9, 0); % 10-element moving average
```

Written by Kenneth S. Paulsen

◆ **nb_num2cell**

```
c = nb_num2cell(d)
```

Description:

Convert rows of a double matrix into a cell array.

Input:

- d : A N x Q x M double

Output:

- c : A N x 1 cell array, each element stores a 1 x Q x M double.

Written by Kenneth Sætherhagen Paulsen

◆ nb_num2str

s = nb_num2str(d,precision)

Description:

Convert a scalar double to string. If precision is a string this function do the same as the num2str function by MATLAB, if on the other hand it is a integer it does not collapse big numbers to e^(something). E.g.

```
num2str(6000.354,2) = 6e+03  
nb_num2str(6000.354,2) = 6000.35
```

Input:

- d : Scalar double.

- precision : A char (see num2str) or an integer with the number of decimals to include.

Output:

- s : A char

Written by Kenneth Sætherhagen Paulsen

◆ nb_percentiles

p = nb_percentiles(d,perc,dim)

Description:

Take the percentile over the given dimension of the data of the object.

Uses mid-point interpolation.

Input:

```
- d      : A double with dimension less than or equal to 3.  
- perc  : A double with the wanted percentiles of the data. E.g.  
          50 (median), or [5,15,25,35,50,65,75,85,95]. These values must  
          be integers!  
- dim   : The dimension to calculate the percentiles over. Default is 1.
```

Output:

```
- p      : A double with the percentiles.
```

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_reduceCov**

```
par = nb_reduceCov(C)
```

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_rlag**

```
xout = nb_rlag(xin,t,len)
```

Description:

Rolling lag operator

Input:

```
- xin  : A double  
- t    : The number of lags  
- len  : The number of periods of the output
```

Output:

```
- xout : A double
```

Examples:

```
xout = nb_rlag([1;2;1;2],4,12)
```

Written by Kenneth S. Paulsen

◆ **nb_rrref**

```
A = nb_rrref(X)
```

Description:

This method produces the row echelon form of a matrix A.

Input:

- A : A NxM double matrix.

Output:

- A : A NxM double matrix.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_seasonalDummy**

```
dummy = nb_seasonalDummy(T,frequency,type)
```

Description:

Create seasonal dummies

Input:

- T : Number of periods. As an integer.
- frequency : Either 4 (quarterly) or 12 (monthly)
- type : 'uncentered' or 'centered'

Output:

- dummy : The dummy variables as T x frequency-1

Written by Kenneth Sæterhagen Paulsen

◆ **nb_sgrowth**

```
out = nb_sgrowth(data,horizon)
```

Description:

Calculates smoothed 12 or 6 months growth.

$$\begin{aligned}x(*) &= ((x_{13}+x_{14}+x_{15})/(x_1+x_2+x_3)-1)*100 \\x(*) &= ((x_7+x_8+x_9)/(x_1+x_2+x_3)-1)*100\end{aligned}$$

Note: Works only for monthly data.

Based on Anne Sofie Jores data transformation code.

Input:

- data : As a double r*c*p matrix with monthly observations.
- horizon : Either 1 or 2, depending on the frequency you want your growth to be calculated over. 1 is annual and 2 is semi-annual.

Output:

- out : A r*c*p double matrix.

See also:

[nb_ts.sgrowth](#)

Written by Tobias Ingebrigtsen

◆ **nb_slag**

xlag = nb_slag(xin, lags)

Description:

Creates a matrix of nObs x total number of lags. I.e.
[x1(t-1), x1(t-2), ... x1(t-max([lags{1}]), x2(t-1), ...,
x2(t-max([lags{1}]))]

Input:

- xin : A nObs x nVar double.
- lags : A 1 x nVar cell array. Each element must be a vector of integers with the lags to be made for the corresponding column of xin.

Output:

- xlag : A nObs x total number of lags double.

Examples:

```
xin = rand(20,2);
xlag = nb_slag(xin,{1,[1,2]})
```

Written by Kenneth S. Paulsen

◆ **nb_spline**

X = nb_spline(X)

Description:

Fill in for in-sample missing values of a time-series using cubic spline.

Input:

- X : A T x 1 double storing the data of the time-series.

Output:

- X : A T x 1 double storing the balanced dataset of the time-series.

See also:

[nb_ts.rebalance](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_splitSample**

splitted = nb_splitSample(data,nSteps)

Description:

Split a sample into a matrix with size nSteps x nvar x nobs. Will append nan values for the last observations.

Input:

- data : A nobs x nvar double.
- nSteps : The number of period of the splitted data.

Output:

- splitted : A nSteps x nvar x nobs double.

Examples:

splitted = nb_splitSample(rand(20,2),4)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_subAvg**

xout = nb_subAvg(xin,k)
xout = nb_subAvg(xin,k,w)

Description:

- Will for the last k periods (including the present) calculate the cumulative (weighted) sum and then divide by the amount of periods, which gives you the (weighted) average over those periods.

Input:

- xin : A double of size r*s*p.
- k : Lets you choose what frequency you want to calculate the cumulative average over. As a double. E.g. 4 if you have quarterly data and want to calculate the average over the last 4 quarters.
- w : Weights applies to the k periods to sum over. Default is equal weights! As a double vector with length k.

Output:

- xout : A double.

Examples:

```
xin = rand(10,1);
xout = nb_subAvg(xin,4)
xout = nb_subAvg(xin,4,[0.1,0.4,0.4,0.1])
```

See also:

[nb_subSum](#)

Written by Tobias Ingebrigtsen

◆ nb_subSum

```
xout = nb_subSum(xin,k)
xout = nb_subSum(xin,k,w)
```

Description:

- Calculates the cumulative sum over the last k periods (including the present period).

Input:

- xin : A double of size r*s*p
- k : Lets you choose what frequency you want to calculate the cumulative sum over. As a double. E.g. 4 if you have quarterly data and want to calculate the sum over the last 4 quarters.
- w : Weights applies to the k periods to sum over. Default is equal weights! As a double vector with length k.

Output:

- xout : A double.

Examples:

```
xin = ones(10,1);
xout = nb_subSum(xin,4)
xout = nb_subSum(xin,4,[0.1,0.4,0.4,0.1])
```

See also:

[nb_subAvg](#)

Written by Tobias Ingebrigtsen

◆ **nb_trimr**

```
z = nb_trimr(x,n1,n2)
```

Description:

Strip the rows before and included n1 and the rows after and included n2 of a matrix x.

Note: Modeled after Gauss trimr function

Input:

- x : A double matrix.
- n1 : See above. A scalar.
- n2 : See above. A scalar.

Output:

- z : The stripped matrix as a double.

Written by Kenneth S. Paulsen

◆ **nb_undiff**

```
dout = nb_undiff(DX,initialValues,periods)
```

Description:

Inverse of the diff function

Input:

```
- DX : A double with the differenced data

- initialValues : A scalar or a double with the initial
    values of the indicies. Must be of the same
    size as the number of variables of the
    DX input. And size(initialValues,1) == periods

- periods : The number of periods the initial series
    has been taken diff over.
```

Output:

```
- dout : A double
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_undiffnan**

```
dout = nb_undiffnan(din,t,periods)
```

Description:

Inverse of the diff function, but is robust for trailing and leading NaNs. It also checks whether theres any NaNs in between observations in the double, and returns an error if it is the case.

Input:

```
- din : A double matrix with dimensions [r,c,p].
- t : A double matrix with dimensions [r,c,p].
- periods : The number of periods the din has been growthed over.
```

Output:

```
- dout : A double matrix with dimensions [r,c,p].
```

Examples:

```
x = rand(3,3,3);
y = nan(1,3,3);
t = [y;x;y];

nb_undiffnan(t,100)
```

See also:

[nb_undiff](#), [nb_diff](#), [growth](#), [igrowth](#), [igrowthnan](#)

Written by Kenneth Sæterhagen Paulsen

◆ **pcn**

```
dout = pcn(din,t)
```

See also:

[growth](#), [growth](#), [ipcn](#)

Written by Kenneth S. Paulsen

◆ **q2y**

```
xout = q2y(xin)
```

Description:

Transform quarterly to annual, using sum over the last 4 periods.

See also:

[growth](#)

Written by Kenneth Sæterhagen Paulsen

◆ **steady_state**

```
x = steady_state(x)
```

Description:

Return the same as in. Used by nb_dsgen when checking the steady-state of a model.

See also:

[nb_dsgen.checkSteadyState](#)

Written by Kenneth Sæterhagen Paulsen

◆ **steady_state_first**

```
x = steady_state_first(x)
```

Description:

Return the same as in. Used by nb_dsgen when checking the steady-state of a model.

See also:

[nb_dsgen.checkSteadyState](#)

Written by Kenneth Sæterhagen Paulsen

25.10 IO functions

- nb_any2File
 - nb_copyfile
 - nb_joinPath
 - nb_loadFromUserpath
 - nb_makeWritable
 - nb_parseOneOptional
 - nb_pasteFromClipboard
 - nb_save2Userpath
 - nb_userpath
 - nb_writeError
 - nb_copyToClipboard
 - nb_error
 - nb_load
 - nb_loadWithName
 - nb_parseInputs
 - nb_parseOneOptionalSingle
 - nb_pathCorrect
 - nb_syncFolders
 - nb_wrapped
-

◆ nb_any2File

```
nb_any2File(filename, input)
nb_any2File(filename, input, nestedLevel, numPrecision)
```

Description:

Write a NBTOOLBOX objects to file.

Input:

- filename : Either a string with the name of the output file or file identifier.
- input : A variable (not the name of the variable)
- nestedLevel : Level of nesting of cell arrays and struct.

Output:

- code : A cellstr with the code.

Written by Kenneth Sæterhagen Paulsen

◆ nb_copyToClipboard

```
nb_copyToClipboard(x)
nb_copyToClipboard(x, dec, hSep, vSep)
```

Description:

This function puts a string, cell array, or numerical array into the clipboard.

Input:

- x : One of:
 - > A char array.
 - > A numerical array.
 - > A cell array of the above.Limited to 2-dimensional arrays.
- dec : A character that indicates the decimal separator. Default is the period ('.') .
- hSep : A character that indicates how to split horizontal elements in a given row vector. Default is char(9)
- vSep : A character that indicates how to split horizontal elements in a given row vector. Default char(10)

Examples:

```
nb_copyToClipboard('Test')
nb_copyToClipboard(char('Test','2'))
nb_copyToClipboard(rand(3,3))
```

See also:

[nb_pasteFromClipboard](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_copyfile

`nb_copyfile(from,to,format)`

Description:

Copy a file to a new location

Input:

- from : The full path name of the copied file. As a string.
- to : The full path name of the new location of the file.
(Including the file name). As a string.
- format : > 'text' : A normal file is copied.
> otherwise : A .mat file is copied.

Examples:

```
nb_copyfile('C:\docs\test.txt', 'C:\newfile.txt', 'text')
```

Written by Kenneth S. Paulsen

◆ nb_error

```
nb_error(message,Err)
```

Description:

Add extra message to a Matlab MException object.

Input:

- message : A string
- Err : Matlab MException object

Written by Kenneth Sæterhagen Paulsen

◆ nb_joinPath

```
out = nb_joinPath(p,f,e)
```

Description:

- Appends a backslash, \, to the end of a cellstring with the path name and then concatenates all the file components into a single cellstring.

Input:

- p,f,e : The different file parts. As cellstrings.

Output:

- out : A single cellstring with all the filepart concatenated.

Examples:

```
p = 'Matlab\B\test'  
f = 'fcsttest'  
e = 'xlsx'  
  
nb_joinPath(p,f,e)  
  
ans = 'Matlab\B\test\fcsttest.xlsx'
```

See also:

Written by Kenneth Sæterhagen Paulsen

◆ nb_load

```
l = nb_load(filename)
l = nb_load(filename,index)
```

Description:

Load .mat file using the MATLAB function load, then select the index of the variable to load. Default is 1.

Input:

- filename : Name of file to load. See the MATLAB load function.
- index : The load function return a struct, where the fields are the separate variables that are stored to the .mat file. The index input refer to the index value of the variable you want to sign the output. Default is 1.

Output:

- l : The loaded variable from the .mat file.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_loadFromUserpath**

```
variable      = nb_loadFromUserpath(filename)
[variable,err] = nb_loadFromUserpath(filename)
```

Description:

Save a variable to the subfolder nb_GUI of the userpath folder.

Input:

- filename : Name of the created file.

Output:

- variable : The variable load from the file.
- err : If nargout > 1 this give an non-empty char if an error occurred.

See also:

[nb_save2Userpath](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_loadWithName**

```
nb_loadWithName(fileName,givenName,workSpace)
```

Description:

Load one (first) MATLAB variable from a .mat file with the same name as the filename or the givenName.

Input:

- fileName : Name of the .mat file to load the MATLAB variable from.
- givenName : If you want to assign the loaded MATLAB variable with another name then the fileName input this option may be used.
- workSpace : Which workspace to load the variable into. Either 'base' (default) or 'caller'.

Output:

- A MATLAB variable stored in the base workspace with the wanted name.

Written by Kenneth SÃ¶therhagen Paulsen

◆ nb_makeWritable

```
c = nb_makeWritable(c)
```

Description:

Make a cellstr or char ready to be written to a file.

Input:

- c : Either a n x 1 cellstr or a n x m char.

Output:

- c : A n x 1 cellstr.

See also:

[nb_cellstr2file](#)

Written by Kenneth SÃ¶therhagen Paulsen

◆ nb_parseInputs

```
[inputs,message,index] = nb_parseInputs(varargin)
```

Description:

Parse inputs.

Input:

- mfile : The name of the function that calls the parser.

- default : A cell with the following:

```
{'inputName1', inputValue1, @isnumeric;...
  'inputName2', inputValue2, {@isa,'nb_ts'},...
  'inputName2', inputValue2, {@isnumeric,'&&',@isscalar}}
```

Can also be a cell array with 4 columns. In the 4th column you can include a function_handle that will be called when the corresponding property is being set. If the element is anything else than a function_handle, it will not be done anything. The function_handle need to take 2 inputs. The input name and the input value (In that order).

- varargin : The input name, input value pairs from the user.

Output:

- inputs : A struct with the parsed options.

- message : Error message. If empty, there where no problems

- index : An index of the inputs that are set by the varargin input. A logical vector with size size(default,1), where the matching element in default is true if set.

Written by Kenneth Sætherhagen Paulsen

◆ nb_parseOneOptional

```
[value,rest] = nb_parseOneOptional(name,default,varargin)
```

Description:

Parse one optional input at the time.

Input:

- name : Name of the optional input as a string.

- default : The default value of the optional input.

- varargin : All the optional inputs.

Output:

- value : Either the found or default value.
- rest : Rest of the optional inputs.

Written by Kenneth Sætherhagen Paulsen

◆ nb_parseOneOptionalSingle

```
[value,rest] = nb_parseOneOptionalSingle(name,ifNotGiven,ifGiven,varargin)
```

Description:

Parse one optional input at the time.

Input:

- name : Name of the optional input as a string.
- ifNotGiven : The value return if name is not found in varargin.
- ifGiven : The value return if name is found in varargin.
- varargin : All the optional inputs.

Output:

- value : Either ifNotGiven or ifGiven.
- rest : Rest of the optional inputs.

Written by Kenneth Sætherhagen Paulsen

◆ nb_pasteFromClipboard

```
x = nb_pasteFromClipboard
x = nb_pasteFromClipboard(dec,hSep,vSep,convert)
```

Description

Does the inverse operation of nb_copyToClipboard, but it can also interpret copied ranges from a excel spreadsheet.

If it cannot figure out the input, a one line char will be returned.

Input:

- dec : A character that indicates the decimal separator. Default is the period ('.') .
- hSep : A character that indicates how to split horizontal elements in a given row vector. Default is char(9)

- vSep : A character that indicates how to split horizontal elements in a given row vector. Default char(10)
- convert : Try to convert to numerical or not. true or false. Default is true. If all elements are numbers a double matrix is returned, otherwise a cell array.

Output :

x : A char, cell or double depending on the input.

Examples:

```
nb_copyToClipboard(char('Test','2'));
p1 = nb_pasteFromClipboard

nb_copyToClipboard(rand(3,3));
p2 = nb_pasteFromClipboard

nb_copyToClipboard('Test');
p3 = nb_pasteFromClipboard

c = {'Var1','Var2';2,2};
nb_copyToClipboard(c);
p4 = nb_pasteFromClipboard
```

See also:

[nb_copyToClipboard](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_pathCorrect**

path = nb_pathCorrect (path)

Description:

Replace \ with \\ in path name.

Inputs:

- path : A 1xN char or cellstr.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_save2Userpath**

nb_save2Userpath(filename,variable)

Description:

Save a variable to the subfolder nb_GUI of the userpath folder.

Input:

- filename : Name of the created file.
- variable : The variable to save to the file.

See also:

[nb_loadFromUserpath](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_syncFolders

`nb_syncFolders(folder1, folder2)`

Description:

Delete the folders and files in folder folder2, which are not in the folder folder1

Input:

- folder1 : A string with the path of the folder.
- folder2 : A string with the path of the folder.

Output:

- deleted : A cellstr with the deleted files and folders

Written by Kenneth Sæterhagen Paulsen

◆ nb_userpath

`uPath = nb_userpath(type)`

Description:

Get userpath. If the path return by the MATLAB function userpath is empty the pwd is used instead.

Input:

- type : A string with either
 - > 'normal' : userpath or pwd (without semicolon at the end). (default)
 - > 'gui' : Same as normal but added \nb_GUI at the end

Output:

- uPath : A string. For more see inputs

Written by Kenneth SÃ¶therhagen Paulsen

◆ nb_wrapped

```
wrapped = nb_wrapped(string,max)
```

Description:

Wrap line at spaces, where each line of the wrapped text is a maximum number of chars.

Input:

- string : A one line char.
- max : Max number of char in one line.

Output:

- wrapped : A cellstr.

Examples:

```
s = 'This is a line that must be wrapped. Can you help me?'  
c = nb_wrapped(s, 40)
```

Written by Tobias Ingebrigtsen

◆ nb_writeError

```
nb_writeError(Err,fileToWrite,extra)
```

Description:

Write error message to file.

Input:

- Err : A MException object.
- fileToWrite : Either a file identifier or the name of the file to write the error message to. As a string.
- extra : A N x M char with extra information on the error. Will be added first in the report written to the file.

Written by Kenneth SÃ¶therhagen Paulsen

25.11 Struct functions

- nb_addFields
- nb_collapsStruct
- nb_defaultField
- nb_isfield
- nb_keepFields
- nb_rmfield
- nb_struct
- nb_struct2cellarray
- nb_structEqual
- nb_structStrcat
- nb_structStrrep
- nb_structcat
- nb_structdepcat
- nb_uncollapsStruct

■ nb_struct

Go to: [Properties](#) | [Methods](#)

A handle representation of a MATLAB structure (struct).

All the function that works on a struct will also work on a nb_struct.

Differences are:

- Initializing with two inputs, see below.
- Assign a multivalued struct (e.g. with size 1 x 4) with a cell;

```
s = nb_struct(4,{'t'})  
s.t = {'t','d','e','f'};  
s.t  
  
ans = {'t','d','e','f'}  
  
s(4).t = {{'g'}};
```

- Caution: Don't set the fields of this type of struct to other structs, as it is not supported to assign thoose struct. Use instead a normal MATLAB struct!

Superclasses:

`matlab.mixin.Copyable, handle`

Constructor:

`nb_struct(varargin)`

Input:

- Same input as for a MATLAB struct with one exception:

If the first input is a integer and the second is a cellstr, it will create a nb_struct with size varargin{1} and with fields given by the cellstr varargin{2}.

Output:

- `obj` : A nb_struct object.

Examples:

```
st1 = nb_struct();
```

```
st2 = nb_struct('test',{},'test2',{});  
st3 = nb_struct([]) % Empty
```

See also:

[struct](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

• [s](#)

- [s](#) ↑

A normal MATLAB struct.

Methods:

• [arrayfun](#)

• [isempty](#)

• [nb_defaultField](#)

• [orderfields](#)

• [struct2table](#)

• [fieldnames](#)

• [isfield](#)

• [nb_isfield](#)

• [rmfield](#)

• [structfun](#)

• [getfield](#)

• [isstruct](#)

• [nb_keepFields](#)

• [size](#)

• [vertcat](#)

• [horzcat](#)

• [nb_addFields](#)

• [nb_rmfield](#)

• [struct2cell](#)

► **arrayfun** ↑

```
obj = arrayfun(obj,field)
```

Description:

This method is not supported by this class.

Written by Kenneth Sætherhagen Paulsen

► **fieldnames** ↑

```
fNames = fieldnames(obj)
```

Description:

Get fieldnames of the nb_struct object.

Input:

- obj : An object of class nb_struct.

Output:

- Same output(s) as the fieldnames function of a normal MATLAB struct.

Written by Kenneth Sætherhagen Paulsen

► getfield ↑

```
value = getfield(obj,varargin)
```

Description:

Get a field from the nb_struct object.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the getfield function of a normal MATLAB struct.

Output:

- Same output(s) as the getfield function of a normal MATLAB struct.

Written by Kenneth Sætherhagen Paulsen

► horzcat ↑

```
obj = horzcat(obj,varargin)
```

Description:

Horizontal concatenation of nb_struct objects. [obj,a]

Input:

- obj : An object of class nb_struct.

Optional inputs:

- varargin : Object of class nb_struct.

Output:

- obj : An object of class nb_struct.

See also:

[nb_struct.vertcat](#)

Written by Kenneth Sæterhagen Paulsen

► **isempty** ↑

```
ret = isempty(obj)
```

Description:

Check if the nb_struct object is empty.

Input:

- obj : An object of class nb_struct.

Output:

- Same output as the isempty function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **isfield** ↑

```
value = isfield(obj,varargin)
```

Description:

Is the field name a field of the nb_struct object.

Input:

- obj : An object of class nb_struct.

- Otherwise the same inputs(s) as the isfield function of a normal MATLAB struct.

Output:

- Same output(s) as the isfield function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **isstruct** ↑

```
value = isstruct(obj)
```

Description:

Is the nb_struct object a struct.

Input:

- obj : An object of class nb_struct.

Output:

- value : true

Written by Kenneth Sæterhagen Paulsen

► **nb_addFields** ↑

```
s = nb_addFields(obj, fields)
```

Description:

Add empty fields to a nb_struct.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the nb_addFields function of a normal MATLAB struct.

Output:

- Same output(s) as the nb_addFields function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_defaultField** ↑

```
obj = nb_defaultField(obj,field,default)
```

Description:

Add field with default value if not found found to be a field of an existing nb_struct obj.

Input:

- obj : An object of class nb_struct.
- field : Name of field. As a 1xN char.
- default : Default value assign to field if not found.

Output:

- obj : An object of class nb_struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_isfield** ↑

```
TF = nb_isfield(obj, varargin)
```

Description:

Recursive expansion of isfield.

Input:

- obj : An object of class nb_struct.

Output:

- TF : true if the fieldnames provided are recursive fields of the nb_struct.

Examples:

```
s.fieldLevelOne.fieldLevelTwo = 'valueLevelTwo';
TF = nb_isfield(s, 'fieldLevelOne', 'fieldLevelTwo');
```

See also:

[nb_struct.isfield](#)

Written by Henrik Hortemo Halvorsen

► **nb_keepFields** ↑

```
s = nb_keepFields(obj, fields)
```

Description:

Keep selected fields of nb_struct.

Input:

- obj : A nb_struct object.
- fields : A cellstr.

Output:

- obj : A nb_struct object.

Written by Henrik Hortemo Halvorsen

► **nb_rmffield** ↑

obj = nb_rmffield(obj,field)

Description:

Same as rmfield, but also handle the case where field is not a field of obj.

Input:

- s : A nb_struct object.
- field : A char or cellstr.

Output:

- obj : A nb_struct object.

See also:

[nb_struct.rmfield](#)

Written by Kenneth Sæterhagen Paulsen

► **orderfields** ↑

varargout = orderfields(obj,varargin)

Description:

Order the fields of the nb_struct object.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the orderfields function of a normal MATLAB struct.

Output:

- Same output(s) as the orderfields function of a normal MATLAB struct. Just that the struct is switched to a nb_struct. Remember that nb_struct object is of a handle class.

Written by Kenneth SÃ¶terhagen Paulsen

► **rmfield** ↑

```
obj = rmfield(obj,field)
```

Description:

Remova a field from the nb_struct object.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the rmfield function of a normal MATLAB struct.

Output:

obj : An object of class nb_struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **size** ↑

```
varargout = size(obj,varargin)
```

Description:

Get the size of the nb_struct object.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the size function of a normal MATLAB struct.

Output:

- Same output(s) as the size function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **struct2cell** ↑

```
value = struct2cell(obj,varargin)
```

Description:

Convert a nb_struct object into a cell.

Input:

- obj : An object of class nb_struct.

Output:

- Same output(s) as the struct2cell function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **struct2table** ↑

```
t = struct2table(obj,varargin)
```

Description:

Convert the nb_struct object into a table.

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the struct2table function of a normal MATLAB struct.

Output:

- Same output(s) as the struct2table function of a normal MATLAB struct.

Written by Kenneth Sæterhagen Paulsen

► **structfun** ↑

```
varargout = structfun(obj,varargin)
```

Description:

Evaluate a function on the nb_struct object. For more see the structfun method of a MATLAB struct,

Input:

- obj : An object of class nb_struct.
- Otherwise the same inputs(s) as the structfun function of a normal MATLAB struct.

Output:

- Same output(s) as the structfun function of a normal MATLAB struct.

Written by Kenneth Sætherhagen Paulsen

► **vertcat** ↑

```
obj = vertcat(obj,varargin)
```

Description:

Vertical concatenation of nb_struct objects. [obj;a]

Input:

- obj : An object of class nb_struct.

Optional inputs:

- varargin : Object of class nb_struct.

Output:

- obj : An object of class nb_struct.

See also:

[nb_struct.horzcat](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_addFields**

```
s = nb_addFields(s, fields)
```

Description:

Add empty fields to a struct.

Input:

- s : A struct.
- fields : A cellstr.

Output:

- s : A struct.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_collapseStruct**

```
cs = nb_collapseStruct(s)
```

Description:

Collapse a 1 x N or N x 1 struct into a 1 x 1. If a field differs across N the field will be stored as a 1 x N array of same type as the original field. One exception is if a field is a double and not a scalar, then it will be a double with size N in the dimension of the double + 1.

Caution : This function assumes that a field has the same size and is of the same type in the array dimension of the struct.

Caution : The extra fields N and collapse are added to store original size and collapsing of fields.

Input:

- s : A 1 x N or N x 1 struct.

Output:

- cs : A 1 x 1 struct.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_defaultField**

```
s = nb_defaultField(s,field,default)
```

Description:

Add field with default value if not found found to be a field of an existing struct s.

Input:

- s : A struct.
- field : Name of field. As a 1xN char.
- default : Default value assign to field if not found.

Output:

- s : A struct.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_isfield**

TF = nb_isfield(s, varargin)

Description:

Recursive expansion of isfield.

Input:

- s : A struct.

Output:

- TF : true if the fieldnames provided are recursive fields of the struct.

Examples:

```
s.fieldLevelOne.fieldLevelTwo = 'valueLevelTwo';
TF = nb_isfield(s, 'fieldLevelOne', 'fieldLevelTwo');
```

See also:

[isfield](#)

Written by Henrik Hortemo Halvorsen

◆ **nb_keepFields**

s = nb_keepFields(s, fields)

Description:

Keep selected fields of struct.

Input:

- s : A struct.
- fields : A cellstr.

Output:

- s : A struct.

Written by Henrik Hortemo Halvorsen

◆ **nb_rmffield**

t = nb_rmffield(s,field)

Description:

Same as rmfield, but also handle the case where field is not a field of s.

Input:

- s : A Struct
- field : A char or cellstr.

Output:

- t : A struct.

See also:

[rmfield](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_struct2cellarray**

c = nb_struct2cellarray(s,type)

Description:

Convert struct to cell.

Input:

```
- s : A struct.  
  
- type : > 'array' (default):  
  
    Convert a struct to a cell array. I.e.  
    {'fieldName1',field1,'fieldName2',field2,...}  
  
> 'matrix':  
  
    Convert a struct to a cell array. I.e.  
    {'fieldName1',field1;'fieldName2',field2;...}
```

Output:

```
- c : A cell array
```

See also:

[struct2cell](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_structEqual**

```
ret = nb_structEqual(s1,s2)
```

Description:

Test for equality of two structs.

Input:

```
- t      : A struct or a nb_struct  
  
- r      : A struct or a nb_struct
```

Output:

```
- ret : true or false. They are false if they have different fields  
      and the same fields contains different values. Cell arrays  
      are not tested, and if a field of the struct is a cell false is  
      returned.
```

See also:

[struct](#), [nb_struct](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_structStrcat**

```
[coeff_out] = nb_structStrcat(coeff_in, what, where)
```

Description:

Use this function to add letters in front or at the end of the fieldnames of a struct. It adds 'coeff' at the front as default.

Input:

- coeff_in : The struct in question.
- what : The letters to add. Default is to add 'coeff'.
- where : Where to add the letters: 'front' or 'back'. Default is 'front'.

Output:

coeff_out : A struct with the updated names.

Example:

```
coeff_out = nb_structStrcat(coeff_in,'letters','back')
```

Written by Erling Motzfeldt Kravik

◆ **nb_structStrrep**

```
[coeff_out] = nb_structStrrep(coeff_in, old, new)
```

Description:

Use this function to remove certain letters in the fieldnames of a struct. It removes 'coeff' as default.

Input:

- coeff_in : The struct in question.
- old : The letters to remove. Default is 'coeff'.
- new : The letters to add. Default is ''.

Output:

coeff_out : A struct with the updated names.

Example:

```
coeff_out = nb_structStrcat(coeff_in,'LAMBDA','rho')
```

Written by Erling M. Kravik

◆ nb_structcat

```
s = nb_structcat(t,r)
```

Description:

Concatenated structs. If they have conflicting fields with same fieldname it will give an error (if type is not set to 'first' or 'last').

Input:

- t : A struct or a nb_struct
- r : A struct or a nb_struct
- type : Set to 'first' to use the conflicting fields of the first input, and set to 'last' to use the second input.

Output:

- s : A struct or a nb_struct. If one of the inputs is a nb_struct the the output will be a nb_struct.

See also:

[struct](#), [nb_struct](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_structdepcat

```
s = nb_structdepcat(t,r)
```

Description:

Generalize the [] operator for structs to handle struct with different fields.

```
s1 = struct('test',1,'test2',2);
s2 = struct('test',2);
s = [s1,s2];
```

Will fail, but not

```
s = nb_structdepcat(s1,s2);
```

Input:

- t : A struct or a nb_struct
- r : A struct or a nb_struct

Output:

- s : A struct or a nb_struct. If one of the inputs is a nb_struct the the output will be a nb_struct.

See also:

[struct](#), [nb_struct](#)

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_uncollapsStruct**

cs = nb_uncollapsStruct(s)

Description:

Reverse the operation of nb_collapsStruct.

Input:

- s : A 1 x 1 struct.

Output:

- cs : A 1 x N or N x 1 struct.

Written by Kenneth SÃ¶terhagen Paulsen

25.12 Other functions

- [findjobj](#)
- [nb_List](#)
- [nb_callMethod](#)
- [nb_checkForErrors](#)
- [nb_clock](#)
- [nb_combine](#)
- [nb_conditionalStatement](#)
- [nb_countCombine](#)
- [nb_createLinkToClass](#)
- [nb_default](#)
- [nb_duplicated](#)
- [nb_func2Cellstr](#)
- [nb_getHistPeriodsDuringReporting](#)
- [nb_getTypes](#)
- [nb_intersect](#)
- [nb_isScalarLogical](#)
- [nb_ismemberi](#)
- [nb_rowVector](#)
- [nb_sizeEqual](#)
- [nb_union](#)
- [vec](#)
- [getundoc](#)
- [nb_any2code](#)
- [nb_callMethodWaitbar](#)
- [nb_clearall](#)
- [nb_colVector](#)
- [nb_conditional](#)
- [nb_copy](#)
- [nb_createDispTable](#)
- [nb_createLinkToClassProperty](#)
- [nb_depcat](#)
- [nb_evalExpression](#)
- [nb_getCombinations](#)
- [nb_getOpt](#)
- [nb_getTypesC](#)
- [nb_isLogical](#)
- [nb_isempty](#)
- [nb_parseOptions](#)
- [nb_shuntingYardAlgorithm](#)
- [nb_stateDates](#)
- [nb_when2Notify](#)

■ **nb_List**

Go to: [Properties](#) | [Methods](#)

```
obj = nb_List()
```

Description:

This is a class for storing objects. Using a list structure

Constructor:

```
obj = nb_List()
```

The empty constructor of the nb_List class

See also:

nb_Node

Written by Kenneth Sæterhagen Paulsen

Properties:

- first
- ids
- last

- **first** ↑

The first nb_Node object of the list

- **ids** ↑

A cellstr with the identifiers of the stored objects.

- **last** ↑

The last nb_Node object of the list

Methods:

- add
- get
- getFirst
- getLast
- getNumberOfStoredObjects
- remove
- reset

► **add** ↑

obj = add(obj,aObj,id)

Description:

Add object to the nb_list object

Input:

- obj : An object of class nb_List
- aObj : All inputs are supported
- id : The identifier of the given input. As a string.

Caution : If not given it will be stored with a string with the number of existing objects stored in the object + 1. I.e. if it is stored 2 object in the list, the identifier for the added object will be '3'.

Output:

- obj : An object of class nb_List, where the input aObj are added to the list of stored objects.

Examples:

```
obj.add(aObj);
obj.add(aObj, 'unique identifier');
```

Written by Kenneth S. Paulsen

► **get ↑**

```
storedObj = get(obj,id)
```

Description:

Get a stored object from a list.

Input:

- obj : An object of class nb_List
- id : The identifier of the stored object. As a string.

Output:

- storedObj : The stored object. Could be of any type.

Examples:

```
obj.get('unique identifier');
```

Written by Kenneth S. Paulsen

► **getFirst ↑**

```
firstNode = getFirst(obj)
```

Description:

Get the first node

Input:

- obj : An object of class nb_List

Output:

- firstNode : The first node, as an nb_Node object.

Examples:

```
firstNode = obj.getFirst()
```

Written by Kenneth S. Paulsen

► **getLast** ↑

```
lastNode = getLast(obj)
```

Description:

Get the last node

Input:

- obj : An object of class nb_List

Output:

- lastNode : The last node, as an nb_node object

Examples:

```
lastNode = obj.getLast()
```

Written by Kenneth S. Paulsen

► **getNumberOfStoredObjects** ↑

```
number = getNumberOfStoredObjects(obj)
```

Description:

Get the number of objects stored in the list

Input:

- obj : An object of class nb_List

Output:

- number : The number of stored objects as an integer (double).

Examples:

```
n = obj.getNumberOfStoredObjects()
```

Written by Kenneth S. Paulsen

► **remove** ↑

```
obj = remove(obj,id)
```

Description:

Remove an object from the nb_List object

Input:

- obj : An object of class nb_List
- id : A unique identifier as a string.

Output:

- obj : The nb_List object without the object with the given identifier.

Examples:

```
obj = obj.remove('unique identifier')
```

Written by Kenneth S. Paulsen

► **reset** ↑

```
obj = reset(obj,aObj,id)
```

Description:

Reset an object of the nb_List object

Input:

- obj : An object of class nb_List
- aObj : All inputs are supported
- id : A unique identifier as a string.

Output:

- obj : The nb_List object with the object with the given identifier reset.

Examples:

```
obj = reset(obj,nb_ts(),'data')
```

Written by Kenneth S. Paulsen

■ nb_Node

Go to: [Properties](#) | [Methods](#)

```
obj = nb_Node(element,id)
```

Superclasses:

handle

Description:

A class for storing any type and include it in a list (nb_list object)

Constructor:

```
obj = nb_Node(element,id)
```

Input:

- element : Any type
- id : A string with an identifier.

Output:

- obj : An object of class nb_Node

Examples:

```
obj = nb_Node(2,'A number with value 2');
```

See also:

[nb_list](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- element • id • next • prev
- **element** ↑

The element stored by the node. Can be of any class.

- **id** ↑

The identifier associated with the stored object

- **next** ↑

The next object (when stored in a nb_List object)

- **prev** ↑

The previous object (when stored in a nb_List object)

Methods:

- [getElement](#)
 - [getID](#)
 - [getNext](#)
 - [getPrev](#)
 - [hasNext](#)
 - [isempty](#)
 - [setElement](#)
 - [setNext](#)
 - [setPrev](#)
-

► **getElement** ↑

```
elem = getElement(obj)
```

Description:

Get the elements of an nb_Node object

Input:

- obj : An object of class nb_Node

Output:

- obj : The element of the nb_Node object. Can be of any MATLAB types (objects).

Examples:

```
elem = obj.getElement();
```

Written by Kenneth S. Paulsen

► **getID** ↑

```
id = getID(obj)
```

Description:

Get the id of the object

Input:

- obj : An object of class nb_Node

Output:

- id : The id of the object. As a string.

Examples:

```
obj = obj.getID();
```

Written by Kenneth S. Paulsen

► **getNext ↑**

```
nextObj = getNext(obj)
```

Description:

Get the next object stored in an nb_List object.

Input:

- obj : An object of class nb_Node

Output:

- nextObj : An object of class nb_Node or an empty double [].

Examples:

```
obj = obj.getNext();
```

Written by Kenneth S. Paulsen

► **getPrev ↑**

```
prevObj = getPrev(obj)
```

Description:

Get the previous node stored in an nb_List object

Input:

- obj : An object of class nb_Node

Output:

- prevObj : An object of class nb_Node or an empty double [].

Examples:

```
obj = obj.getPrev();
```

Written by Kenneth S. Paulsen

► **hasNext ↑**

```
obj = setNext(obj,nextObj)
```

Description:

Test for the existence of a next object (when stored in an nb_List object)

Input:

- obj : An object of class nb_Node

Output:

- ret : true (1) if the next object is not empty, otherwise false (0).

Examples:

```
ret = obj.hasNext();
```

Written by Kenneth S. Paulsen

► **isempty ↑**

```
ret = isempty(obj)
```

Description:

A test if the object is empty, returns true if empty

Input:

- obj : An object of class nb_Node

Output:

- ret : true (1) if the object is empty, else false (0)

Examples:

```
ret = objisempty();
```

Written by Kenneth S. Paulsen

► **setElement** ↑

```
obj = setElement(obj,newElement)
```

Description:

Set the element of an object of class nb_Node

Input:

- obj : An object of class nb_Node

- newElement : The new element. Can be of any MATLAB type
(object)

Output:

- obj : An object of class nb_Node with the new element stored

Examples:

```
obj = obj.setElement(newElement);
```

Written by Kenneth S. Paulsen

► **setNext** ↑

```
obj = setNext(obj,nextObj)
```

Description:

Sets the next node (in a list)

Input:

- obj : An object of class nb_Node
- nextObj : The next object (class nb_Node) in an nb_List object

Output:

- obj : An object of class nb_Node with the next property set

Examples:

```
obj = obj.setNext(aObj);
```

Written by Kenneth S. Paulsen

► **setPrev ↑**

```
obj = setPrev(obj,nextObj)
```

Description:

Sets the previous node (in a list)

Input:

- obj : An object of class nb_Node
- nextObj : The previous object (class nb_Node) in an nb_List object

Output:

- obj : An object of class nb_Node with the prev property set

Examples:

```
obj = obj.setPrev(aObj);
```

Written by Kenneth S. Paulsen

◆ **findjobj**

findjobj Find java objects contained within a specified java container or Matlab GUI handle

Syntax:

```
[handles, levels, parentIds, listing] = findobj(container, 'PropertyName', PropValue(s), ...)
```

Input parameters:

container - optional handle to java container uipanel or figure. If unsupplied then current figure is used.
'PropertyName', PropValue - optional list of property pairs (case insensitive). PropName may also be a string containing a regular expression.
'position' - filter results based on those elements that contain the specified X,Y position.
Note: specify a Matlab position (X,Y = pixels from bottom left corner),
'size' - filter results based on those elements that have the specified W,H (in pixels)
'class' - filter results based on those elements that contain the substring (or regular expression).
Note1: filtering is case insensitive and relies on regexp, so you can pass anything.
Note2: '-class' is an undocumented findobj PropName, but only works on Java components.
'property' - filter results based on those elements that possess the specified case-insensitive property.
Note1: passing a property value is possible if the argument following 'property' is in the format of {'propName','propValue'}. Example: FINDOBJ(...,'property','Text','click me')
Note2: partial property names (e.g. 'Tex') are accepted, as long as they are unique.
'depth' - filter results based on specified depth. 0=top-level, Inf=all levels (all components)
'flat' - same as specifying: 'depth',0
'not' - negates the following filter: 'not','class','c' returns all elements EXCEPT those matching the class.
'persist' - persist figure components information, allowing much faster results for repeated calls.
'nomenu' - skip menu processing, for "lean" list of handles & much faster processing.
This option is the default for HG containers but not for figure, Java and uicontrol components.
'print' - display all java elements in a hierarchical list, indented appropriately.
Note1: optional PropValue of element index or handle to java container
Note2: normally this option would be placed last, after all filtering is done.
Note3: output is to the Matlab command window unless the 'listing' (4th) argument is supplied.
'list' - same as 'print'
'debug' - list found component positions in the Command Window

Output parameters:

handles - list of handles to java elements
levels - list of corresponding hierarchy level of the java elements (top=0)
parentIds - list of indexes (in unfiltered handles) of the parent container of the corresponding handles
listing - results of 'print'/'list' options (empty if these options were not specified)

Note: If no output parameter is specified, then an interactive window will be displayed which shows a tree view of all container components, their properties and callbacks.

Examples:

```
findobj;                                % display list of all javaelements of currrent figure in an interactive window
handles = findobj;                        % get list of all java elements of current figure (inc. menu bars)
findobj('print');                         % list all java elements in current figure
findobj('print',6);                       % list all java elements in current figure, contained within 6 levels
handles = findobj(hButton);                % hButton is a matlab button
handles = findobj(gcf,'position',getpixelposition(hButton,1)); % same as above but also includes menu bars
handles = findobj(hButton,'persist');       % same as above, persists between calls
handles = findobj('class','pushbutton');    % get all pushbuttons in current figure
handles = findobj('class','pushbutton','position',123,456); % get all pushbuttons at position 123,456
handles = findobj(gcf,'class','pushbutton','size',23,15);   % get all pushbuttons with size 23x15
handles = findobj('property','Text','not','class','button'); % get all non-button elements
handles = findobj('-property',{'Text','click me'});          % get all elements with Text property
```

Sample usage:

```
hButton = uicontrol('string','click me');
jButton = findobj(hButton,'nomenu');
% or: jButton = findobj('property',{'Text','click me'});
jButton.setFlyOverAppearance(1);
```

```
jButton.setCursor(java.awt.Cursor.getPredefinedCursor(java.awt.Cursor.HAND_CURSOR));
set(jButton,'FocusGainedCallback',@myMatlabFunction); % some 30 callback points available
jButton.get; % list all changeable properties...
```

```
hEditbox = uicontrol('style','edit');
jEditbox = findobj(hEditbox,'nomenu');
jEditbox.setCaretColor(java.awt.Color.red);
jEditbox.KeyTypedCallback = @myCallbackFunc; % many more callbacks where this came from
jEdit.requestFocus;
```

Technical explanation & details:

<http://undocumentedmatlab.com/blog/findobj/>
<http://undocumentedmatlab.com/blog/findobj-gui-display-container-hierarchy/>

Known issues/limitations:

- Cannot currently process multiple container objects - just one at a time
- Initial processing is a bit slow when the figure is laden with many UI components (so be patient)
- Passing a simple container Matlab handle is currently filtered by its position+size: skip
- Matlab uipanels are not implemented as simple java panels, and so they can't be found via findobj
- Labels have a write-only text property in java, so they can't be found using the 'property'

Warning:

This code heavily relies on undocumented and unsupported Matlab functionality.
It works on Matlab 7+, but use at your own risk!

Bugs and suggestions:

Please send to Yair Altman (altmany at gmail dot com)

Change log:

2015-01-12: Differentiate between overlapping controls (for example in different tabs); fixed a few bugs
2014-10-20: Additional fixes for R2014a, R2014b
2014-10-13: Fixes for R2014b
2014-01-04: Minor fix for R2014a; check for newer FEX version up to twice a day only
2013-12-29: Only check for newer FEX version in non-deployed mode; handled case of invisible scroll-panes
2013-10-08: Fixed minor edge case (retrieving multiple scroll-panes)
2013-06-30: Additional fixes for the upcoming HG2
2013-05-15: Fix for the upcoming HG2
2013-02-21: Fixed HG-Java warnings
2013-01-23: Fixed callbacks table grouping & editing bugs; added hidden properties to the table
2013-01-13: Improved callbacks table; fixed tree refresh failure; fixed: tree node-select bug
2012-07-25: Fixes for R2012a as well as some older Matlab releases
2011-12-07: Fixed 'File is empty' messages in compiled apps
2011-11-22: Fix suggested by Ward
2011-02-01: Fixes for R2011a
2010-06-13: Fixes for R2010b; fixed download (m-file => zip-file)
2010-04-21: Minor fix to support combo-boxes (aka drop-down, popup-menu) on Windows
2010-03-17: Important release: Fixes for R2010a, debug listing, objects not found, component
2010-02-04: Forced an EDT redraw before processing; warned if requested handle is invisible
2010-01-18: Found a way to display label text next to the relevant node name
2009-10-28: Fixed uitreenode warning
2009-10-27: Fixed auto-collapse of invisible container nodes; added dynamic tree tooltips
2009-09-30: Fix for Matlab 7.0 as suggested by Oliver W; minor GUI fix (classname font)
2009-08-07: Fixed edge-case of missing JIDE tables
2009-05-24: Added support for future Matlab versions that will not support JavaFrame
2009-05-15: Added sanity checks for axes items
2009-04-28: Added 'debug' input arg; increased size tolerance 1px => 2px
2009-04-23: Fixed location of popupmenus (always 20px high despite what's reported by Matlab)
2009-04-09: Automatic 'nomenu' for uicontrol inputs; significant performance improvement
2009-03-31: Fixed position of some Java components; fixed properties tooltip; fixed node

2009-02-26: Indicated components visibility (& auto-collapse non-visible containers); auto
2009-02-24: Fixed update check; added dedicated labels icon
2009-02-18: Fixed compatibility with old Matlab versions
2009-02-08: Callbacks table fixes; use uiinspect if available; fix update check according
2008-12-17: R2008b compatibility
2008-09-10: Fixed minor bug as per Johnny Smith
2007-11-14: Fixed edge case problem with class properties tooltip; used existing object i
2007-08-15: Fixed object naming relative property priorities; added sanity check for ille
2007-08-03: Fixed minor tagging problems with a few Java sub-classes; displayed UI Class ID
2007-06-15: Fixed problems finding HG components found by J. Wagberg
2007-05-22: Added 'nomenu' option for improved performance; fixed 'export handles' bug; f
2007-04-23: HTMLized classname tooltip; returned top-level figure Frame handle for figure
2007-04-19: Fixed edge case of missing figure; displayed tree hierarchy in interactive GU
2007-04-04: Improved performance; returned full listing results in 4th output arg; enable
2007-03-20: First version posted on the MathWorks file exchange: <a href="http://www.math

See also:

[java](#), [handle](#), [findobj](#), [findall](#), [javaGetHandles](#), [uiinspect](#), (on, the, File, Exchange)

Written by Yair M. Altman

◆ **getundoc**

undocumentedProps = getundoc(arg, skipStandardProps)

Written by Yair Altman

◆ **nb_any2code**

code = nb_any2code(input, varName)

Description:

Convert a MATLAB object to .m code.

Input:

- input : A variable (not the name of the variable)
- varName : Name of the assign variable.
- nestedLevel : Level of nesting of cell arrays and struct.

Output:

- code : A cellstr with the code.

Examples:

See also:

◆ nb_callMethod

```
result = nb_callMethod(obj,meth,classConst,varargin)
```

Description:

Call method meth on the matrix of objects obj.

Input:

- obj : A N x M object of any class.
- meth : The method to call.
- classConst : Class constructor as a function_handle.

Optional input:

- varargin : Given to the method specified by meth.

Output:

- result : A N x M object of class cl.

Written by Kenneth Sæterhagen Paulsen

◆ nb_callMethodWaitbar

```
result = nb_callMethodWaitbar(obj,meth,classConst,waitbar,varargin)
```

Description:

Call method meth on the matrix of objects obj using a waitbar.

Input:

- obj : A N x M object of any class.
- meth : The method to call.
- classConst : Class constructor as a function_handle.
- waitbar : A nb_waitbar5 object.

Optional input:

- varargin : Given to the method specified by meth.

Output:

- result : A N x M object of class cl.

Written by Kenneth Sæterhagen Paulsen

◆ nb_checkForErrors

str = nb_checkForErrors(expression,macro)

Description:

Check for basic errors in an expression to be interpreted by the shunting yard algorithm.

Input:

- expression : A char with the expression to be interpreted.
- macro : Set to true to also handle the NB Toolbox macro processing language as well. Default is false.

Output:

- str : If a non-empty char is returned, the expression cannot be interpreted by the shunting yard algorithm.

See also:

[nb_shuntingYardAlgorithm](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_clearall

nb_clearall()
nb_clearall(false)

Description:

Clear all variables, close all figures and clear command window.

Written by Kenneth Sæterhagen Paulsen

◆ nb_clock

t = nb_clock

Description:

Get the current time on the given format.

Input:

- format : A string with the format:

```
> 'vintagemilliseconds' : 'yyyymmddhhnnssqqq'  
> 'vintagelong' : 'yyyymmddhhnnss'  
> 'vintage' : 'yyyymmddhhnn'  
> 'vintageshort' : 'yyyymmdd'  
> 'gui' : 'Date: dd/mm/yyyy Time: hh:nn'
```

Output:

t : A string.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_colVector**

c = nb_colVector(in)

Description:

Secure a column vector.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_combine**

```
vc = nb_combine(v,n)  
vc = nb_combine(v,n,n2)
```

Description:

Pick n elements from the vector v in all possible ways.

Inspired by a function of Matt Fig.

Reference: <http://mathworld.wolfram.com/BallPicking.html>

Input:

- v : A vector.

- n : The number of picked elements.

- n2 : If 3 inputs is given this function will produce a
nb_countCombine(length(v),n,n2) cell array of all combinations
where n to n2 elements are picked from v.

Output:

- vc : A $n!/k!(n-k)!$ x n matrix, or see input n2.

Written by Kenneth Sæterhagen Paulsen

◆ nb_conditional

```
value = nb_conditional(condition, trueValue, falseValue)
```

Description:

If condition is true return trueValue, or else return falseValue.

Written by Henrik Halvorsen Hortemo and Kenneth Sæterhagen Paulsen

◆ nb_conditionalStatement

```
value = nb_conditional(condition, trueFunc, falseFunc)
```

Description:

If condition is true return output from trueFunc, or else return output from falseFunc.

Written by Henrik Halvorsen Hortemo

◆ nb_copy

```
obj = nb_copy(obj)
```

Description:

Make copy of object. Works on both handle and value objects.

Written by Kenneth Sæterhagen Paulsen

◆ nb_countCombine

```
num = nb_countCombine(nMin, nMax)
```

Description:

This is the number of combinations of N things taken nMin to nMax at a time. I.e. the sum of $n!/k!(n-k)!$ for $k = nMin, \dots, nMax$.

Input:

- n : An integer.
- nMin : An integer.
- nMax : An integer.

Output:

- num : An integer.

See also:

[nchoosek](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_createLinkToClass**

link = nb_createLinkToClass(obj, className)

Description:

Get hyperlink to class doc as a char.

Input:

- obj : Any object.
- className : A one line char.

Output:

- link : A one line char.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_createLinkToClassProperty**

link = nb_createLinkToClassProperty(className, propertyName)

Description:

Get hyperlink to class property or method doc as a char.

Input:

- className : A one line char with the name of the class.
- propertyName : A one line char with the name of the property or method.

Output:

- link : A one line char.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_default**

```
value = nb_default(value, default)
```

Description:

Return default if value is empty.

Input:

- value : Any type.
- default : Any type.

Output:

- value = Either value or default.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_depcat**

```
d = nb_depcat(varargin)
```

Description:

Concatenation of inputs along the 3 dimension.

Input:

- d1 : Any.
- d2 : Any, but same as d1.

Output:

- d : Any, but same type as inputs.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_duplicated**

```
isDup      = nb_duplicated(in)
[isDup,dup] = nb_duplicated(in)
```

Description:

Find the unique duplicated elements of a vector.

Input:

- in : Any vector that are supported by the unique function made by MATLAB.

Output:

- isDup : Return true if any duplicated values are located.
- dup : Return the (unique) located duplicated values.

See also:

[unique](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_evalExpression

```
[out,str] = nb_evalExpression(out,type,nInp)
```

Description:

Evaluate an expression interpreted by the shunting yard algorithm.

Input:

- out : A cell with the out output from nb_shuntingYardAlgorithm interpreted by either nb_getTypes or nb_getTypesC.
- type : A double vector. See the type output from nb_getTypes or nb_getTypesC.
- nInp : A double vector. See the nInp output from the nb_shuntingYardAlgorithm function.

Output:

- out : The evaluated output. Can be of any class (it depend on the out input)
- str : If a non-empty char is returned, the expression cannot be interpreted by the shunting yard algorithm.

See also:

[nb_shuntingYardAlgorithm](#), [nb_getTypes](#), [nb_getTypesC](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_func2Cellstr**

`str = nb_func2Cellstr(func)`

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getCombinations**

`varargout = nb_getCombinations(varargin)`

Description:

Get all possible combinations of the optional inputs. Must be either a cell (1,N) or double (1xN).

Input:

- `nMin` : Min number of elements combined from the first input. If empty 1 is used.
- `nMax` : Max number of elements combined from the first input. If empty `length(varargin{1})` is used
- `varargin{1}` : The main input to max combination of. I.e. the `nMin` and `nMax` is only applicable to this input.
- `varargin{2:end}` : The `varargin{1:ii-1}` are replicated with the size of these inputs to create all combinations.

Output:

- `varargout{1}` = Number of elements of the combinations.
- `varargout(2:end)` = The inputs in all its combinations

Examples:

```
[s,vars,lags] = nb_getCombinations(1,4,{'Var1','Var2','Var3','Var4'},1:10);
[s,vars,lags] = nb_getCombinations(1,3,{'Var1','Var2','Var3','Var4'},1:10);
[s,vars,lags] = nb_getCombinations(1,3,{'Var1','Var2','Var3'},[1,2,3]);
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getHistPeriodsDuringReporting**

```
value = nb_getHistPeriodsDuringReporting()
value = nb_getHistPeriodsDuringReporting(freq)
```

Description:

Get number of historical observations to use during reporting.

Output:

- value : The number of observations.

Written by Kenneth SÃ¶therhagen Paulsen

◆ **nb_getOpt**

```
opt = nb_getOpt()
```

Description:

Get default options given to fminsearch used by the NBTOOLBOX

Written by Kenneth SÃ¶therhagen Paulsen

◆ **nb_getTypes**

```
[out,type] = nb_getTypes(out,variables,data,macro,nInp)
```

Description:

Fill in data to variables in the output from nb_shuntingYardAlgorithm function.

Input:

- out : A cell with the out output from nb_shuntingYardAlgorithm function.
- variables : Same as the variables input to the nb_shuntingYardAlgorithm function.
- data : The values of the variables. Either as a double or another object that can be indexed by (:,ii,:) , where ii is the variables{ii} value.
- macro : Set to true to also handle the NB Toolbox macro processing language as well. Default is false.
- nInp : Number of inputs to each function in out.

Output:

- out : A cell with same length as the out input, where the the names of the variables has been changed with the data of those variables.
- type : A vector needed to evaluate the expression using nb_evalExpression.

See also:

[nb_shuntingYardAlgorithm](#), [nb_getTypesC](#), [nb_evalExpression](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_getTypesC**

[out,type] = nb_getTypesC(out,variables,data)

Description:

Fill in data to variables in the output from nb_shuntingYardAlgorithm function.

Input:

- out : A cell with the out output from nb_shuntingYardAlgorithm function.
- variables : Same as the variables input to the nb_shuntingYardAlgorithm function.
- data : The values of the variables. A cell vector with same length as variables.

Output:

- out : A cell with same length as the out input, where the the names of the variables has been changed with the data of those variables.
- type : A vector needed to evaluate the expression using nb_evalExpression.

See also:

[nb_shuntingYardAlgorithm](#), [nb_getTypes](#), [nb_evalExpression](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_intersect**

c = nb_intersect(varargin)

Description:

Extension of the MATLAB function intersect, i.e. handles more than two inputs to intersect.

Input:

- varargin : Any type of input you can give to the two first inputs to the intersect function.

Output:

- c : The unique elements found in all inputs.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isLogical

```
ret = nb_isLogical(x)
```

Description:

Test if x is a logical scalar. MATLAB <true> and <false> will both return true. As will 0 or 1 even if given as a double.

Input:

- x : Any type

Output:

- bool : true or false.

Written by Per Bjarne Bye

◆ nb_isScalarLogical

```
ret = nb_isScalarLogical(x)
```

Description:

Test if x is a scalar logical.

Input:

- x : Any type

Output:

- ret : true or false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_isempty

ret = nb_isempty(in)

Description:

Same as isempty function, but will return true also for structs without fields.

Input:

- in : Any type of inputs.

Output:

- ret : 1 if empty otherwise 0.

See also:

[isempty](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_ismemberi

varargout = nb_ismemberi(varargin)

Description:

Case insensitive ismember

Input:

See ismember

Output:

See ismember

See also:

[ismember](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_parseOptions

```
options = nb_parseOptions(varargin)
```

Description:

Takes structs and key/value-pairs as inputs and returns a struct with lowercase keys

Written by Henrik Halvorsen Hortemo

◆ nb_rowVector

```
c = nb_rowVector(in)
```

Description:

Secure a row vector.

Written by Kenneth Sætherhagen Paulsen

◆ nb_shuntingYardAlgorithm

```
[str,out,nInp,stack,nInpStack,prec] = nb_shuntingYardAlgorithm(...  
    expression,variables,macro)  
[str,out,nInp,stack,nInpStack,prec] = nb_shuntingYardAlgorithm(...  
    expression,variables,macro,out,stack,prec,nInp,nInpStack,last)
```

Description:

MATLAB implementation of the shunting yard algorithm for interpreting a mathematical expression.

Input:

- expression : The expression to interpret. As a 1 x n char.
- variables : The list of variables of the function that the expression consists of.
- macro : Set to true to also handle the NB Toolbox macro processing language as well. Default is false.

The rest of the inputs is used internally in this function.

Output:

- str : If a non-empty char is returned, the expression cannot be interpreted by the shunting yard algorithm.
- out : A cell with the interpretation of the expression.
- nInp : Number of inputs to each function.

See also:

[nb_eval](#), [nb_checkForErrors](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_sizeEqual**

```
ret = sizeEqual(input, shouldBeSize)
```

Description:

Test if the input is of a given size

Input:

- input : Any object
- shouldBeSize : A 1 x n double. Where n must match the potential output size of size(input). If some elements are nan those dimensions are unrestricted.

Output:

- ret : True if the size match shouldBeSize. If the input shouldBeSize does not match the output from size(input) ret is returned as false.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_stateDates**

```
out = nb_stateDates(startYear, endYear, d, freq)
```

Description:

A function to find the release date of variables, given that there is a general rule (day number) which the variable is released. The function controls for Norwegian holidays. If the release date (given by the rule) is on a holiday and/or weekend, it will return the day which is closest.

Input:

- startYear : As an nb_date object or as a string.
- endYear : As an nb_date object or as a string.
- d : The day number where the state variable is published. As a double.
- freq : The frequency of the observations you want. Either 4 or 12. As a double.
- laglead : (Optional) Sets the lag/lead interval of the test.

Output:

- out : A numPeriods*1 nb_day object.

Examples:

```
out = nb_stateDates('2010','2018',10,12)  
out = nb_stateDates('2010','2018',10,4)  
out = nb_stateDates(nb_year('2010'),nb_day('2018M1D10'),10,4)
```

See also:

[nb_easter](#)

Written by Tobias Ingebrigtsen

◆ **nb_union**

c = nb_union(varargin)

Description:

Extension of the MATLAB function union, i.e. handles more than two inputs to union.

Input:

- varargin : Any type of input you can give to the two first inputs to the union function.

Output:

- c : The elements found in some inputs.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_when2Notify**

note = nb_when2Notify(number)

Description:

This function governs how often the waitbars used in the NBTOOLBOX notify the status

Written by Kenneth Sæterhagen Paulsen

◆ vec

```
out = vec(in)
```

Description:

The vector operator. Same as in(:)

Input:

- in : Any array.

Output:

- out : Any array.

Written by Kenneth Sæterhagen Paulsen

25.13 Calendars

- nb_allCalendar
- nb_currentCalendar
- nb_lastCalendar
- nb_manualCalendar
- nb_numDaysCalendar

■ nb_allCalendar

Go to: [Properties](#) | [Methods](#)

An object that will provide the all calendar, i.e. return all runs of a model.

Superclasses:

nb_calendar

Constructor:

```
obj = nb_allCalendar()
```

See also:

[nb_currentCalendar](#), [nb_MPRCalendar](#), [nb_lastCalendar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:**Methods:**

- doOneModel
- getCalendar
- getContextIndex
- getDefaultStart
- getName
- getStartOfCalendar
- getSubClasses
- loadobj
- saveobj
- selectCalendar
- selectRecursiveCalendar
- shrinkCalendar
- struct
- unstruct

► **doOneModel** ↑

```
[index,isMatch] = nb_calendar.doOneModel(model,calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- model : The model of interest. As a nb_model_vintages object.
 - or
 - A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.
- calendar : Full calendar as a N x 1 double.

Output:

- index : The index for the model that matches the provided calendar, as a 1 x N numerical array, where N is the length of the calendar.
- isMatch : A 1 x N logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getCalendar** ↑

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- obj : An object of a subclass of the nb_calendar class.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- modelGroup : A vector of objects of class

```
nb_model_forecast_vintages or a cellstr where
each element is on the format 'yyyymmdd' or
'yyyymmddhhnnss'.
```

- doRecursive : If the modelGroup input is a scalar
nb_model_group_vintages object you may want to
get the calender of the children instead of the
object itself. In this case this input must be
set to true. Default is true.
- fromResults : Give true to take the contexts from results
property instead of the forecastOutput property.

This is only supported when modelGroup is an
array of nb_model_vintages objects.

Output:

- calendar : The calendar for the selected window, as a
N x 1 double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getContextIndex** ↑

```
[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)
```

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.
- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar
dates. Here Q <= M, and is only lower if some of the calendar
dates does not match with any contexts.
- locC : A 1 x M logical. A element is true, if the calendar dates
found a matching context.

See also:

[nb_allCalendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getDefaultStart** ↑

```
start = nb_calendar.getDefaultStart(modelGroup, doRecursive, fromResults)
```

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getName** ↑

```
name = getName(obj)
```

Description:

Get calendar name as one line char

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getStartOfCalendar** ↑

```
contextsStart = nb_calendar.getStartOfCalendar(modelGroup, doRecursive, ...  
fromResults)
```

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- contextsStart : A cellstr array with same length as the modelGroup input, each element stores the first context of the that particular model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of NB Toolbox.

Output:

- classes : A cellstr with the subclasses.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

Load object from .mat

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

Inherited from superclass NB_CALENDAR

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
    doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **selectRecursiveCalendar ↑**

[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is the end date of each recursive window.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► shrinkCalendar ↑

```
[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)
```

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► struct ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► unstruct ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

■ nb_calendar

Go to: [Properties](#) | [Methods](#)

A superclass of the different calendars that can be used to construct time dependent averaging of models in nb_model_group_vintages.constructWeights.

See also:

[nb_model_group_vintages.constructWeights](#)
[nb_model_group_vintages.combineForecast](#)
[nb_allCalendar](#)
[nb_currentCalendar](#)
[nb_lastCalendar](#)
[nb_manualCalendar](#)
[nb_MPRCalendar](#)
[nb_MPRTCutoffCalendar](#)
[nb_numDaysCalendar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

Methods:

- [doOneModel](#)
- [getCalendar](#)
- [getContextIndex](#)
- [getDefaultStart](#)
- [getName](#)
- [getStartOfCalendar](#)
- [getSubClasses](#)
- [loadobj](#)
- [saveobj](#)
- [selectCalendar](#)
- [selectRecursiveCalendar](#)
- [shrinkCalendar](#)
- [struct](#)
- [unstruct](#)

► **doOneModel** ↑

```
[index,isMatch] = nb_calendar.doOneModel(model,calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- model : The model of interest. As a nb_model_vintages object.

or

A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.

- calendar : Full calendar as a N x 1 double.

Output:

- index : The index for the model that matches the provided calendar, as a 1 x N numerical array, where N is the length of the calendar.
- isMatch : A 1 x N logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

► **getCalendar** ↑

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- obj : An object of a subclass of the nb_calendar class.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- modelGroup : A vector of objects of class nb_model_forecast_vintages or a cellstr where each element is on the format 'yyyymmdd' or 'yyyymmdhhnnss'.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- calendar : The calendar for the selected window, as a N x 1 double.

Written by Kenneth Sæterhagen Paulsen

► **getContextIndex** ↑

```
[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)
```

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.
- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar dates. Here Q <= M, and is only lower if some of the calendar dates does not match with any contexts.
- locC : A 1 x M logical. A element is true, if the calendar dates found a matching context.

See also:

[nb_calendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth Sæterhagen Paulsen

► **getDefaultStart** ↑

```
start = nb_calendar.getDefaultStart(modelGroup,doRecursive,fromResults)
```

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth SÃ¶terhagen Paulsen

► getName ↑

```
name = getName(obj)
```

Description:

Get calendar name as one line char

Written by Kenneth SÃ¶terhagen Paulsen

► getStartOfCalendar ↑

```
contextsStart = nb_calendar.getStartOfCalendar(modelGroup, doRecursive, ...  
fromResults)
```

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- contextsStart : A cellstr array with same length as the modelGroup input, each element stores the first context of the that particular model.

Written by Kenneth SÃ¶terhagen Paulsen

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of NB Toolbox.

Output:

- classes : A cellstr with the subclasses.

Written by Kenneth Sæterhagen Paulsen

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

Load object from .mat

Written by Kenneth Sæterhagen Paulsen

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

► **selectRecursiveCalendar ↑**

```
[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)
```

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is

the end date of each recursive window.

- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

► **shrinkCalendar ↑**

[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth Sæterhagen Paulsen

■ **nb_currentCalendar**

Go to: [Properties](#) | [Methods](#)

An object that will provide the current calendar.

Superclasses:

`nb_calendar`

Constructor:

```
obj = nb_currentCalendar(frequency)
```

Input:

- `frequency` : The frequency of the provided kalendar.
 - > 1 : Yearly, i.e. current date each year back in time.
 - > 2 : Semi-annually, i.e. add to each half-year back in time the current number of days since the start of the current half-year implied by todays date.
 - > 4 : Quarterly, i.e. same as for semi-annually, but for quarters instead.
 - > 12: Monthly, i.e. same as for semi-annually, but for months instead.
- Default is yearly (1).

See also:

`nb_MPRCalendar`, `nb_numDaysCalendar`

Written by Kenneth Sæterhagen Paulsen

Properties:

- `frequency`

- **frequency** ↑

The frequency of the current calendar.

Methods:

- `doOneModel`
- `getCalendar`
- `getContextIndex`
- `getDefaultStart`
- `getName`
- `getStartOfCalendar`
- `getSubClasses`
- `loadobj`
- `saveobj`
- `selectCalendar`
- `selectRecursiveCalendar`
- `shrinkCalendar`
- `struct`
- `unstruct`

► **doOneModel** ↑

```
[index, isMatch] = nb_calendar.doOneModel(model, calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- `model` : The model of interest. As a `nb_model_vintages` object.
 - or
 - A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.
- `calendar` : Full calendar as a $N \times 1$ double.

Output:

- `index` : The index for the model that matches the provided calendar, as a $1 \times N$ numerical array, where N is the length of the calendar.
- `isMatch` : A $1 \times N$ logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_CALENDAR`

► **getCalendar** ↑

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- `obj` : An object of a subclass of the `nb_calendar` class.
- `start` : Start date of calendar window, as a `nb_day` object.
- `finish` : End date of calendar window, as a `nb_day` object.
- `modelGroup` : A vector of objects of class
 `nb_model_forecast_vintages` or a `cellstr` where
 each element is on the format '`yyyymmdd`' or
 '`yyyymmddhhnnss`'.
- `doRecursive` : If the `modelGroup` input is a scalar
 `nb_model_group_vintages` object you may want to
 get the calendar of the children instead of the
 object itself. In this case this input must be
 set to true. Default is true.
- `fromResults` : Give true to take the contexts from results
 property instead of the `forecastOutput` property.

This is only supported when `modelGroup` is an array of `nb_model_vintages` objects.

Output:

- `calendar` : The calendar for the selected window, as a `N x 1 double`.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_CALENDAR`

► **getContextIndex** ↑

```
[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)
```

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.
- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar dates. Here Q <= M, and is only lower if some of the calendar dates does not match with any contexts.
- locC : A 1 x M logical. A element is true, if the calendar dates found a matching context.

See also:

[nb_currentCalendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► getDefaultStart ↑

start = nb_calendar.getDefaultStart(modelGroup, doRecursive, fromResults)

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getName** ↑

```
name = getName(obj)
```

Description:

Get calendar name as one line char

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getStartOfCalendar** ↑

```
contextsStart = nb_calendar.getStartOfCalendar(modelGroup, doRecursive, ...  
fromResults)
```

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- contextsStart : A cellstr array with same length as the modelGroup input, each element stores the first context of the that particular model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of NB Toolbox.

Output:

- classes : A cellstr with the subclasses.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

Load object from .mat

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

Inherited from superclass NB_CALENDAR

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► [selectRecursiveCalendar](#) ↑

[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is the end date of each recursive window.
- doRecursive : If the modelGroup input is a scalar

nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **shrinkCalendar** ↑

[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **unstruct** ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

■ **nb_lastCalendar**

Go to: [Properties](#) | [Methods](#)

An object that will provide last calendar, i.e. return the last runs of a model (group).

Superclasses:

nb_calendar

Constructor:

```
obj = nb_lastCalendar()
```

See also:

[nb_currentCalendar](#), [nb_MPRLCalendar](#), [nb_allCalendar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

Methods:

- [doOneModel](#)
- [getCalendar](#)
- [getContextIndex](#)
- [getDefaultStart](#)
- [getName](#)
- [getStartOfCalendar](#)
- [getSubClasses](#)
- [loadobj](#)
- [saveobj](#)
- [selectCalendar](#)
- [selectRecursiveCalendar](#)
- [shrinkCalendar](#)
- [struct](#)
- [unstruct](#)

► **doOneModel** ↑

```
[index,isMatch] = nb_calendar.doOneModel(model,calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- model : The model of interest. As a nb_model_vintages object.
 - or
 - A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.
- calendar : Full calendar as a N x 1 double.

Output:

- index : The index for the model that matches the provided calendar, as a 1 x N numerical array, where N is the length of the calendar.
- isMatch : A 1 x N logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getCalendar ↑**

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- obj : An object of a subclass of the nb_calendar class.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- modelGroup : A vector of objects of class nb_model_forecast_vintages or a cellstr where each element is on the format 'yyyymmdd' or 'yyyymmddhhnnss'.

- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.

- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- calendar : The calendar for the selected window, as a N x 1 double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getContextIndex** ↑

```
[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)
```

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.

- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar dates. Here Q <= M, and is only lower if some of the calendar dates does not match with any contexts.

- locC : A 1 x M logical. A element is true, if the calendar dates found a matching context.

See also:

[nb_lastCalendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getDefaultStart** ↑

```
start = nb_calendar.getDefaultStart(modelGroup, doRecursive, fromResults)
```

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getName** ↑

```
name = getName(obj)
```

Description:

Get calendar name as one line char

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getStartOfCalendar** ↑

```
contextsStart = nb_calendar.getStartOfCalendar(modelGroup, doRecursive, ...  
fromResults)
```

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- contextsStart : A cellstr array with same length as the modelGroup input, each element stores the first context of the that particular model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of NB Toolbox.

Output:

- classes : A cellstr with the subclasses.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

```
Load object from .mat

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR
```

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

Inherited from superclass NB_CALENDAR

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
    doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **selectRecursiveCalendar ↑**

[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is the end date of each recursive window.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► shrinkCalendar ↑

```
[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)
```

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► struct ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► unstruct ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

■ nb_manualCalendar

Go to: [Properties](#) | [Methods](#)

An object that will provide a manually provided calendar.

Superclasses:

nb_calendar

Constructor:

```
obj = nb_manualCalendar(calendar)
```

Input:

- calendar : The manual calendar provided by the user. Must be a cellstr where all elements are on the format 'yyyymmdd', e.g. {'20120101','20130101'}.

See also:

[nb_currentCalendar](#), [nb_getMPRVintages4](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [calendar](#)

- **calendar** ↑

The calendar provided by the user.

Methods:

- | | | |
|-----------------------------------|---|--------------------------------------|
| • doOneModel | • getCalendar | • getContextIndex |
| • getDefaultStart | • getName | • getStartOfCalendar |
| • getSubClasses | • loadobj | • saveobj |
| • selectCalendar | • selectRecursiveCalendar | • shrinkCalendar |
| • struct | • unstruct | |

► **doOneModel** ↑

```
[index,isMatch] = nb_calendar.doOneModel(model,calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- model : The model of interest. As a nb_model_vintages object.
 - or
 - A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.
- calendar : Full calendar as a N x 1 double.

Output:

- index : The index for the model that matches the provided calendar, as a 1 x N numerical array, where N is the length of the calendar.
- isMatch : A 1 x N logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► getCalendar ↑

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- obj : An object of a subclass of the nb_calendar class.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- modelGroup : A vector of objects of class nb_model_forecast_vintages or a cellstr where each element is on the format 'yyyymmdd' or 'yyyymmddhhnnss'.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- calendar : The calendar for the selected window, as a N x 1 double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getContextIndex** ↑

```
[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)
```

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.
- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar dates. Here Q <= M, and is only lower if some of the calendar dates does not match with any contexts.
- locC : A 1 x M logical. A element is true, if the calendar dates found a matching context.

See also:

[nb_manualCalendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getDefaultStart** ↑

```
start = nb_calendar.getDefaultStart(modelGroup,doRecursive,fromResults)
```

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth SÃ!terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getName** ↑

name = getName(obj)

Description:

Get calendar name as one line char

Written by Kenneth SÃ!terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getStartOfCalendar** ↑

contextsStart = nb_calendar.getStartOfCalendar(modelGroup,doRecursive,...
fromResults)

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

```
- modelGroup : A vector of objects of class nb_model_forecast_vintages.  
  
- doRecursive : If the modelGroup input is a scalar  
    nb_model_group_vintages object you may want to  
    get the calender of the children instead of the  
    object itself. In this case this input must be  
    set to true. Default is true.  
  
- fromResults : Give true to take the contexts from results  
    property instead of the forecastOutput property.  
  
    This is only supported when modelGroup is an  
    array of nb_model_vintages objects.
```

Output:

```
- contextsStart : A cellstr array with same length as the modelGroup  
    input, each element stores the first context of the  
    that particular model.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of
NB Toolbox.

Output:

```
- classes : A cellstr with the subclasses.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

Load object from .mat

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

Inherited from superclass NB_CALENDAR

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
    doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **selectRecursiveCalendar** ↑

```
[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)
```

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is the end date of each recursive window.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **shrinkCalendar** ↑

```
[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)
```

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **unstruct** ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

■ **nb_numDaysCalendar**

Go to: [Properties](#) | [Methods](#)

An object that will provide the a calendar with X days after the end of each month, quarter, etc. E.g. if numDays is set to 10, and frequency is set to 4, you will get 10.01.2000, 10.04.2000, ...

Superclasses:

[nb_calendar](#)

Constructor:

obj = nb_numDaysCalendar(numDays, frequency)

Input:

- numDays : Number of days into the period. The frequency of the period will decide what is allowed. Default is 1.
- frequency : The frequency of the provided kalendar.
 - > 1 : Yearly, i.e. each year back in time.
 - > 2 : Semi-annually, i.e. each half-year back in time.
 - > 4 : Quarterly, i.e. each quarter back in time.
 - > 12: Monthly, i.e. each month back in time.
 - > 52: Weekly, i.e. each week back in time.Default is yearly (1).

See also:

[nb_currentCalendar](#), [nb_MPRLCalendar](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [frequency](#)
- [numDays](#)
- [frequency](#) ↑

The frequency of the current calendar.

- [numDays](#) ↑

Number of days into the period.

Methods:

- [doOneModel](#)
- [getCalendar](#)
- [getContextIndex](#)
- [getDefaultStart](#)
- [getName](#)
- [getStartOfCalendar](#)
- [getSubClasses](#)
- [loadobj](#)
- [saveobj](#)
- [selectCalendar](#)
- [selectRecursiveCalendar](#)
- [shrinkCalendar](#)
- [struct](#)
- [unstruct](#)

► [doOneModel](#) ↑

```
[index,isMatch] = nb_calendar.doOneModel(model,calendar)
```

Description:

Shrink calendar to a provided window.

Input:

- model : The model of interest. As a nb_model_vintages object.
 - or
 - A cellstr with the contexts of a model. Each element on the format 'yyyymmdd'.
- calendar : Full calendar as a N x 1 double.

Output:

- index : The index for the model that matches the provided calendar, as a 1 x N numerical array, where N is the length of the calendar.
- isMatch : A 1 x N logical array, if false it means that the model doesn't provide a forecast at the given date, where N is the length of the calendar.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getCalendar ↑**

```
calendar = getCalendar(obj,start,finish,modelGroup,doRecursive)
```

Description:

Get calendar.

Input:

- obj : An object of a subclass of the nb_calendar class.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- modelGroup : A vector of objects of class nb_model_forecast_vintages or a cellstr where each element is on the format 'yyyymmdd' or 'yyyymmddhhnnss'.

- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- calendar : The calendar for the selected window, as a N x 1 double.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getContextIndex** ↑

[indC,locC] = nb_calendar.getContextIndex(forecastContexts,calendar)

Description:

Get index of the contexts that match a set of calendar dates.

Input:

- forecastContexts : A N x 1 or 1 x N double with the contexts.
- contexts : A 1 x M or M x 1 double with the calendar dates.

Output:

- indC : A 1 x Q index of the contexts that match a set of calendar dates. Here Q <= M, and is only lower if some of the calendar dates does not match with any contexts.
- locC : A 1 x M logical. A element is true, if the calendar dates found a matching context.

See also:

[nb_numDaysCalendar.doOneModel](#), [nb_modelDataSource.update](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getDefaultStart** ↑

```
start = nb_calendar.getDefaultStart(modelGroup, doRecursive, fromResults)
```

Description:

Get default start of calendar if the modelGroup is provide.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- start : The default start date of the calendar. As a nb_day object. Is empty ('') if modelGroup is empty.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getName** ↑

```
name = getName(obj)
```

Description:

Get calendar name as one line char

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getStartOfCalendar** ↑

```
contextsStart = nb_calendar.getStartOfCalendar(modelGroup, doRecursive, ...  
fromResults)
```

Description:

Get first context of a set of nb_model_forecast_vintages, or the children of a scalar nb_model_group_vintages object.

Input:

- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calender of the children instead of the object itself. In this case this input must be set to true. Default is true.
- fromResults : Give true to take the contexts from results property instead of the forecastOutput property.

This is only supported when modelGroup is an array of nb_model_vintages objects.

Output:

- contextsStart : A cellstr array with same length as the modelGroup input, each element stores the first context of the that particular model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **getSubClasses** ↑

```
classes = nb_calendar.getSubClasses()
```

Description:

Get all available subclasses of the nb_calendar for your version of NB Toolbox.

Output:

- classes : A cellstr with the subclasses.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **loadobj** ↑

```
obj = nb_calendar.loadobj(s)
```

Description:

```
Load object from .mat

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR
```

► **saveobj** ↑

```
s = saveobj(obj)
```

Description:

Save object to .mat. Called by the save method.

Written by Kenneth S. Paulsen

Inherited from superclass NB_CALENDAR

► **selectCalendar** ↑

```
[index,calendar] = selectCalendar(obj,modelGroup,start,finish,...  
    doRecursive,fromResults)
```

Description:

Select the forecasts that matches the supplied calendar for each model/model group.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : End date of calendar window, as a nb_day object.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array. Each element stores a numerical array with the indexes of the forecasts of the given calendar.
- calendar : The calendar used for the given model.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **selectRecursiveCalendar ↑**

[index,calendar] = selectRecursiveCalendar(obj,modelGroup,start,finish,doRecursive)

Description:

Select the forecast that matches the supplied calendar for each model/model group recursively.

Input:

- obj : An object of a subclass of the nb_calendar class.
- modelGroup : A vector of objects of class nb_model_forecast_vintages.
- start : Start date of calendar window, as a nb_day object.
- finish : A 1 x nContext array of nb_day objects. Each element is the end date of each recursive window.
- doRecursive : If the modelGroup input is a scalar nb_model_group_vintages object you may want to get the calendar of the children instead of the object itself. In this case this input must be set to true. Default is true.

Output:

- index : The index for each model that matches the provided calendar, as a 1 x nModel cell array, each element consisting of 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a numerical array with the indexes of the forecasts of the given calendar of a given recursive evaluation period.
- calendar : A 1 x nRec cell array, where nRec is the number of recursive periods. Each element of these again stores a the calendar of each recursive period.

See also:

[nb_model_group_vintages.constructWeights](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALENDAR

► **shrinkCalendar** ↑

```
[calendar,ind] = nb_calendar.shrinkCalendar(calendar,start,finish)
```

Description:

Shrink calendar to a provided window.

Input:

- calendar : Full calendar as a N x 1 double.
- start : A nb_day object, or empty.
- finish : A nb_day object, or empty.

Output:

- calendar : The calendar of the provided window.
- ind : A logical of size N x 1. True for each elements kept.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to a struct.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

► **unstruct** ↑

```
obj = nb_calendar.unstruct(s)
```

Description:

Convert struct to a object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALENDAR

25.14 Diagnostics

- nb_LMVARTest
- nb_adf
- nb_autocorr
- nb_autocorrMat2
- nb_autocov
- nb_autocovMat2
- nb_baiPerronTestStatistic
- nb_bootstrap
- nb_breuschGodfreyTest
- nb_calcRoots
- nb_chowTest
- nb_copulaBootstrap
- nb_durbinWatson
- nb_egcoint
- nb_grangerTest
- nb_infoCriterion
- nb_lagLengthSelection
- nb_normalityTest
- nb_olsLikelihood
- nb_phillipsPerron
- nb_rSquared
- nb_sarganHansen
- nb_smirnovTest
- nb_trapaniNFactorTest
- nb_varFTest
- nb_SERegression
- nb_archTest
- nb_autocorrMat
- nb_autocorrTest
- nb_autocovMat
- nb_automaticModelSelection
- nb_blockBootstrap
- nb_boxPierceQTest
- nb_breuschPaganTest
- nb_cfsTest
- nb_cointTest
- nb_cucconiTest
- nb_durbinWuHausman
- nb_fTest
- nb_hornParallelAnalysis
- nb_jcoint
- nb_ljungBoxTest
- nb_olsConfInt
- nb_parcorr
- nb_quantileWaldTest
- nb_restrictedFTest
- nb_shockDecomp
- nb_spearmanTest
- nb_unitRootTest
- nb_whiteTest

■ nb_LMVARTestStatistic

Go to: [Properties](#) | [Methods](#)

A class for doing the LM autocorrelation test on a nb_var object.

See the function nb_LMVARTest for more on this test.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_LMVARTestStatistic(model,varargin)
```

Input:

- model : An nb_LMVARTestStatistic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_LMVARTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
- help
- print
- set
- template

► **doTest** ↑

```
obj = doTest(obj)
```

Description:

Do LM test for autocorrelated residuals. Results are stored in the property results.

Input:

- obj : An object of class nb_LMVARTestStatistic.

Output:

- obj : An object of class nb_LMVARTestStatistic.

Written by Kenneth SÃ¶therhagen Paulsen

► **help** ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to nb_LMVARTestStatistic

Input:

- obj : A nb_LMVARTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶therhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_LMVARTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth SÃ¶therhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

```
options = nb_LMVARTestStatistic.template()
```

Description:

Construct a struct which must be provided to the nb_LMVARTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ nb_archTestStatistic

Go to: [Properties](#) | [Methods](#)

A class for doing ARCH test on an nb_singleEq object.

Superclasses:

nb_test_generic

Constructor:

obj = nb_archTestStatistic(model,varargin)

Input:

- model : A nb_model_generic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_archTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
 - help
 - print
 - set
 - template
-

► **doTest** ↑

```
obj = doTest(obj)
```

Description:

Do ARCH-test. Results are stored in the property results.

Input:

- obj : An object of class nb_archTestStatistic.

Output:

- obj : An object of class nb_archTestStatistic.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to nb_archTestStatistic

Input:

- obj : A nb_archTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth Sæterhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_archTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

```
options = nb_archTestStatistic.template()
```

Description:

Construct a struct which must be provided to the nb_archTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_autocorrTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing autocorrelation test on a nb_singleEq object.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_autocorrTestStatistic(model,varargin)
```

Input:

- model : A nb_model_generic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_autocorrTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
 - help
 - print
 - set
 - template
-

► **doTest** ↑

obj = doTest(obj)

Description:

Do autocorrelation-test. Results are stored in the property results.

Input:

- obj : An object of class nb_autocorrTestStatistic.

Output:

- obj : An object of class nb_autocorrTestStatistic.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

help = help(~,option)

Description:

A method to give some basic instructions regarding input to nb_autocorrTestStatistic

Input:

- obj : A nb_autocorrTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶terhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_autocorrTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth SÃ¶terhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.
Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template ↑**

options = nb_autocorrTestStatistic.template()

Description:

Construct a struct which must be provided to the nb_autocorrTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Återhagen Paulsen

■ **nb_baiPerronTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing Bai and Perron multiple structural break test.

See:

Bai, Jushan and Pierre Perron (1998): "Estimating and Testing Linear Models with Multiple Structural Changes," *Econometrica*, vol 66, 47-78 and

Bai, Jushan and Pierre Perron (2003): "Computation and Analysis of Multiple Structural Change Models," *Journal of Applied Econometrics*, 18, 1-22.

Constructor:

```
obj = nb_baiPerronTestStatistic(data,varargin)
```

Input:

- data : An object of class nb_ts. Can only consist of one page (dataset).

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : A nb_baiPerronTestStatistic object.

See also:

[baiPerron.pbreak](#)

Written by Kenneth Åström Paulsen

Properties:

- data • options • results

- **data** ↑

The test time-series as an nb_ts object.

- **options** ↑

The test options provided to the nb_adf, nb_phillipsPeron or the nb_kpss functions.

- **results** ↑

A struct storing the results of the test.

Methods:

- doTest • help • print • set • template

► **doTest** ↑

```
obj = doTest(obj)
```

Description:

Do the Bai-Perron structural break test. Results are stored in the property results.

Input:

- obj : An object of class nb_baiPerronTestStatistic.

Output:

- obj : An object of class nb_baiPerronTestStatistic.

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_baiPerronTestStatistic.help(option)
```

Description:

Get the help on the different test options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_chowTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_baiPerronTestStatistic objects.

Input:

- obj : A vector of nb_baiPerronTestStatistic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the nb_baiPerronTestStatistic.template method.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_baiPerronTestStatistic objects.

Written by Kenneth Sæterhagen Paulsen

► **template** ↑

options = nb_baiPerronTestStatistic.template()

Description:

Construct a struct which must be provided to the nb_baiPerronTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ nb_breuschGodfreyTestStatistic

Go to: [Properties](#) | [Methods](#)

A class for doing Breusch-Godfrey test on a nb_singleEq object.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_breuschGodfreyTestStatistic(model,varargin)
```

Input:

- model : A nb_model_generic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_breuschGodfreyTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
 - help
 - print
 - set
 - template
-

► **doTest** ↑

```
obj = doTest(obj)
```

Description:

Do Breusch-Godfrey autocorrelation test. Results are stored in the property results.

Input:

- obj : An object of class nb_breuschGodfreyTestStatistic.

Output:

- obj : An object of class nb_breuschGodfreyTestStatistic.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to nb_breuschGodfreyTestStatistic

Input:

- obj : A nb_breuschGodfreyTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth Sæterhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_breuschGodfreyTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth SÃ¶terhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

```
options = nb_breuschGodfreyTestStatistic.template()
```

Description:

Construct a struct which must be provided to the nb_breuschGodfreyTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_breuschPaganTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing the Breusch-Pagan test on an nb_singleEq object.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_breuschPaganTestStatistic(model,varargin)
```

Input:

- model : A nb_model_generic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_breuschPaganTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
 - help
 - print
 - set
 - template
-

► **doTest** ↑

obj = doTest(obj)

Description:

Do Breusch-Pagan heteroscedasticity test. Results are stored in the property results.

Input:

- obj : An object of class nb_breuschPaganTestStatistic.

Output:

- obj : An object of class nb_breuschPaganTestStatistic.

Written by Kenneth SÅtterhagen Paulsen

► **help** ↑

help = help(~,option)

Description:

A method to give some basic instructions regarding input to nb_breuschPaganTestStatistic

Input:

- obj : A nb_breuschPaganTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶terhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_breuschPaganTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth SÃ¶terhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.
Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template ↑**

options = nb_breuschPaganTestStatistic.template()

Description:

Construct a struct which must be provided to the nb_breuschPaganTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Återhagen Paulsen

■ **nb_chowTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing chow test on an nb_singleEq object.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_chowTestStatistic(model,varargin)

Input:

- model : A nb_model_generic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a
    property of the object or one of the fields of the
    options property. (See the static template method
    for more.)
```

Output:

```
- obj : An nb_chowTestStatistic
```

Written by Kenneth Åström Paulsen

Properties:

- model • options • results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest • help • plot • print • set • template

► **doTest** ↑

```
obj = doTest(obj)
```

Description:

Do Chow-test. Results are stored in the property results.

Input:

```
- obj : An object of class nb_chowTestStatistic.
```

Output:

- obj : An object of class nb_chowTestStatistic.

Written by Kenneth Sæterhagen Paulsen

► help ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to nb_chowTestStatistic

Input:

- obj : A nb_chowTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth Sæterhagen Paulsen

► plot ↑

```
plotter = plot(obj)
```

Description:

Plot recursive chow test p-values

Input:

- obj : An object of class nb_chowTestStatistic.

Output:

- plotter : An object of class nb_graph_ts. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

► print ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_chowTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

```
options = nb_chowTestStatistic.template()
```

Description:

Construct a struct which must be provided to the nb_chowTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_cointTest**

Go to: [Properties](#) | [Methods](#)

A class for cointegration testing of time-series stored in an nb_ts object.

Constructor:

```
obj = nb_cointTest(data,type,varargin)
```

Input:

- data : An object of class nb_ts. Can only consist of one page (dataset).

- type : A string with either:

> 'eg' : Engle-Granger single equation test.
(nb_egcoint)

> 'jo' : Johansen system test. (nb_jcoint)

Optional input:

- Depends on the type input. For information look at the documentation of the function in parentheses above.

Output:

- obj : An nb_cointTest test object.

See also:

[nb_egcoint](#), [nb_jcoint](#), [nb_ts](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- **data**
- **estimationEq**
- **options**
- **results**
- **transformation**
- **type**

- **data** ↑

The test time-series as an nb_ts object.

- **estimationEq** ↑

A 1 x n vector of nb_olsEstimator object storing the estimation results (by OLS).

- **options** ↑

The test options provided to the nb_jcoint or nb_egcoint functions.

- **results** ↑

A struct storing the results of the test. Will depend on the type of test. See the documentation of the functions nb_egcoint and nb_jcoint for more on this structure.

- **transformation** ↑

The transformation of the input data. Either:

```
> 'level'      : Do nothing (default)
> 'firstDiff'  : First difference
> 'secondDiff' : Second difference
```

- **type** ↑

The type of test.

Methods:

- **print**
- **set**

► **print** ↑

```
res = print(obj,precision)
```

Description:

Get the test results as a char.

Input:

- obj : An nb_unitRootTest object.

Output:

- results : A char with the test results.

Written by Kenneth Sæterhagen Paulsen

► set ↑

set(obj,varargin)

Description:

Set the properties of an nb_cointTest object

Input:

- obj : An object of class nb_cointTest
- varargin : ..., 'propertyName', 'propertyValue', ...

Examples:

obj.set('propertyName', 'propertyValue', ...)

Written by Kenneth S. Paulsen

■ nb_durbinWuHausmanStatistic

Go to: [Properties](#) | [Methods](#)

A class for doing the Durbin-Wu-Hausman test for exogeneity on a nb_singleEq object

Superclasses:

nb_test_generic

Constructor:

obj = nb_durbinWuHausmanStatistic(model, varargin)

Input:

- model : A nb_singleEq object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_durbinWuHausmanStatistic

Written by Kenneth Sæterhagen Paulsen

Properties:

- model • options • results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest • help • print • set • template

► **doTest** ↑

obj = doTest(obj)

Description:

Do the Durbin-Wu-Hausman test. Results are stored in the property results.

Input:

- obj : An object of class nb_durbinWuHausmanStatistic. See template for the options of the test.

Output:

- obj : An object of class nb_durbinWuHausmanStatistic.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to
nb_durbinWuHausmanStatistic

Input:

- obj : A nb_archTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth Sæterhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_durbinWuHausmanStatistic.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template ↑**

options = nb_durbinWuHausmanStatistic.template()

Description:

Construct a struct which must be provided to the nb_durbinWuHausmanStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_fTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing F-test on a nb_singleEq object

Superclasses:

nb_test_generic

Constructor:

obj = nb_fTestStatistic(model, varargin)

Input:

- model : A nb_singleEq object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_fTestStatistic

Written by Kenneth Sæterhagen Paulsen

Properties:

- model • options • results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest • help • print • set • template

► **doTest** ↑

obj = doTest(obj)

Description:

Do f-test. Results are stored in the property results.

Input:

- obj : An object of class nb_fTestStatistics, where the restrictions to test is stored in options.A and options.c. ($A \cdot \beta = c$)

Output:

- obj : An object of class nb_fTestStatistics.

Written by Kenneth SÃ¶terhagen Paulsen

► help ↑

```
help = help(~,option)
```

Description:

A method to give some basic instructions regarding input to nb_fTestStatistic

Input:

- obj : A nb_fTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶terhagen Paulsen

► print ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_fTestStatistics.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

obj = set(obj,varargin)

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

options = nb_fTestStatistic.template()

Description:

Construct a struct which must be provided to the nb_fTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ nb_ljungBoxTestStatistic

Go to: [Properties](#) | [Methods](#)

A class for doing the Ljung-Box test on a nb_var object.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_ljungBoxTestStatistic(model,varargin)
```

Input:

- model : An nb_ljungBoxTestStatistic object.

Optional input:

- varargin : 'inputName', inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_ljungBoxTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- [model ↑](#)

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- [options ↑](#)

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest
 - help
 - print
 - set
 - template
-

► **doTest** ↑

obj = doTest(obj)

Description:

Do Ljung-Box test. Results are stored in the property results.

Input:

- obj : An object of class nb_ljungBoxTestStatistic.

Output:

- obj : An object of class nb_ljungBoxTestStatistic.

Written by Kenneth SÃ¶therhagen Paulsen

► **help** ↑

help = help(~,option)

Description:

A method to give some basic instructions regarding input to nb_ljungBoxTestStatistic

Input:

- obj : A nb_ljungBoxTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶therhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_ljungBoxTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template** ↑

```
options = nb_lljungBoxTestStatistic.template()
```

Description:

Construct a struct which must be provided to the nb_archTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_sarganHansenTestStatistic**

Go to: [Properties](#) | [Methods](#)

A class for doing the Sargan-Hansen overidentification test.

See documentation of DAG for more on this test.

Superclasses:

nb_test_generic

Constructor:

```
obj = nb_sarganHansenTestStatistic(model,varargin)
```

Input:

- model : An nb_sarganHansenTestStatistic object.

Optional input:

- varargin : 'inputName',inputValue pairs. Will either set a property of the object or one of the fields of the options property. (See the static template method for more.)

Output:

- obj : An nb_sarganHansenTestStatistic object

Written by Kenneth Sæterhagen Paulsen

Properties:

- model • options • results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

Inherited from superclass NB_TEST_GENERIC

- **options** ↑

A struct storing all the options for the test.

Inherited from superclass NB_TEST_GENERIC

- **results** ↑

A struct storing the test results

Inherited from superclass NB_TEST_GENERIC

Methods:

- doTest • help • print • set • template

► **doTest** ↑

obj = doTest(obj)

Description:

Do Sargan-Hansen test. Results are stored in the property results.

Input:

- obj : An object of class nb_sarganHansenTestStatistic.

Output:

- obj : An object of class nb_sarganHansenTestStatistic.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

help = help(~,option)

Description:

A method to give some basic instructions regarding input to
nb_sarganHansenTestStatistic

Input:

- obj : A nb_sarganHansenTestStatistic object
- option : A string with the property to look up.

Output:

- help : A string with the help text.

Written by Kenneth SÃ¶terhagen Paulsen

► **print** ↑

```
res = print(obj)
```

Description:

Print test results.

Input:

- obj : An object of class nb_sarganHansenTestStatistic.

Output:

- res : A string with the test results.

Written by Kenneth SÃ¶terhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TEST_GENERIC

► **template ↑**

options = nb_sarganHansenTestStatistic.template()

Description:

Construct a struct which must be provided to the nb_sarganHansenTestStatistic class constructor.

This structure provided the user the possibility to set different test options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

■ **nb_test_generic**

Go to: [Properties](#) | [Methods](#)

A overhead class to do classical test on a nb_model_generic or a nb_estimator class.

Constructor:

Not possible to initialize an nb_test_generic object. The class is abstract.

Written by Kenneth Sæterhagen Paulsen

Properties:

- model
- options
- results

- **model** ↑

The nb_model_generic or nb_estimator object to be tested.

- **options** ↑

A struct storing all the options for the test.

- **results** ↑

A struct storing the test results

Methods:

- set
-

► **set** ↑

obj = set(obj,varargin)

Description:

Sets the properties of the nb_test_generic objects.

Caution : It will set the fields of the properties of the object.

Input:

- obj : A vector of nb_test_generic objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_test_generic objects.

Written by Kenneth Sæterhagen Paulsen

■ nb_unitRootTest

Go to: [Properties](#) | [Methods](#)

A class for unit root testing of time-series stored in an nb_ts object.

Constructor:

```
obj = nb_unitRootTest(data,type,varargin)
```

Input:

- data : An object of class nb_ts. Can only consist of one page (dataset).
- type : A string with either:
 - > 'adf' : Augmented Dickey-Fuller Test. (nb_adf)
Default.
 - > 'pp' : Phillips-Peron Test. (nb_phillipsPeron)
 - > 'kpss' : Kwiatkowski, Phillips, Schmidt and Shin test. (nb_kpss)

Optional input:

- Depends on the type input. For information look at the documentation of the function in parentheses above.

Output:

- obj : An nb_unitRootTest test object.

See also:

[nb_adf](#), [nb_phillipsPeron](#), [nb_kpss](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- data • estimationEq • options • results • transformation
- type
- **data** ↑

The test time-series as an nb_ts object.

- **estimationEq** ↑

A 1 x numberOfVariables struct storing the estimation results (by OLS).

- **options** ↑

The test options provided to the nb_adf, nb_phillipsPeron or the nb_kpss functions.

- **results** [↑](#)

A struct storing the results of the test. Will depend on the type of test. See the documentation of the functions nb_adf, nb_phillipsPeron and nb_kpss for more on this structure.

Caution : This structure will be of size 1 x
numberOfVariables. I.e. obj.results(1) stores
the test results for variable 1 in the nb_ts
object.

- **transformation** [↑](#)

The transformation of the input data. Either:

```
> 'level'      : Do nothing (default)
> 'firstDiff'  : First difference
> 'secondDiff' : Second difference
```

- **type** [↑](#)

The type of test.

Methods:

- [print](#)
 - [set](#)
-

► **print** [↑](#)

```
res = print(obj,precision)
```

Description:

Get the test results as a char.

Input:

- obj : An nb_unitRootTest object.

Output:

- results : A char with the test results.

Written by Kenneth Sæterhagen Paulsen

► **set** [↑](#)

```
set(obj,varargin)
```

Description:

Set the properties of an nb_unitRootTest object

Input:

- obj : An object of class nb_unitRootTest
- varargin : ..., 'propertyName', 'propertyValue', ...

Examples:

```
obj.set('propertyName', 'propertyValue', ...)
```

Written by Kenneth S. Paulsen

◆ **nb_LMVARTest**

```
[stat,pval] = nb_LMVARTest(residual,X,j)
```

Description:

Test for autocorrelation of residuals of a VAR model using the LM test. See section 4.3.3 of Juselius (2006).

Auxiliary regression:

```
residual(t) = A*X(t) + B*residual_(t-j) + u(t)
```

Input:

- residual : A nobs x neq double with the residuals.
- X : The models regressors, as a nobs x nxvar double. Should not include the constant!
- j : The lag (of the residual) to include in the auxiliary regression.

Output:

- stat : The test-statistics.
- pval : The p-value.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_SERegression**

```
[SERegression,sumOfSqRes] = nb_SERegression(residual,numCoeff)
```

Description:

Calculates the Standard Error of the Regression and the Sum-of-Squared Residuals.

Input:

- residual : The residuals from the ols regression. As a nobs x neqs double.
- numCoeff : The number of estimated coefficients of the model. As a 1 x neqs double.

Output:

- SERegression : Standard Error of the Regression. As a double.

Calculated using the formula:

$$s = \sqrt{\text{residual}' * \text{residual} / (T - \text{numCoeff})}$$

- sumOfSqRes : Sum-of-Squared Residuals

$$S = \text{residual}' * \text{residual}$$

Written by Kenneth Åkerhagen Paulsen

◆ **nb_adf**

```
results = nb_adf(y)
[results,output] = nb_adf(y,varargin)
```

Description:

Augmented Dickey-Fuller tests for unit root. I.e. testing the null hypothesis of a unit root.

Input:

- y : A nb_ts object or a double (nobs x nvars).

Optional input:

- 'nLags' : Sets the number of lags of the estimated model to base the test upon. Default is 0.
- 'maxLags' : Sets the maximum number of lags used by the lag length criteria. Default is 10.
- 'lagLengthCrit' : Lag length selection criterion. Either:

```

> 'aic'    : Akaike information criterion
> 'aicc'   : Corrected Akaike information
               criterion
> 'maic'   : Modified Akaike information
               criterion
> 'sic'    : Schwarz information criterion
> 'msic'   : Modified Schwarz information
               criterion
> 'hqc'    : Hannan-Quinn information criterion
> 'mhqc'   : Modified Hannan-Quinn information
               criterion
> ''       : Manually set lag length.

- 'model'      : Either ('ar' is default):
> 'ar'     :  $y(t) = \beta_1 y(t-1) + \beta_2 (1-L)y(t-1) + \dots + \beta_{p+1} (1-L)y(t-p) + e(t)$ 
> 'ard'    :  $y(t) = \beta_1 + \beta_2 y(t-1) + \beta_3 (1-L)y(t-1) + \dots + \beta_{p+2} (1-L)y(t-p) + e(t)$ 
> 'ts'     :  $y(t) = \beta_1 + \beta_2 t + \beta_3 y(t-1) + \beta_4 (1-L)y(t-1) + \dots + \beta_{p+3} (1-L)y(t-p) + e(t)$ 

- 'method'    : One of;
               > 'standard' : The standard ADF test statistics.
               > 'badf'     : The backward ADF statistic sequence.
               > 'bsadf'   : The backward sup ADF statistic
                  sequence.
               > 'gsadf'   : The generalized sup ADF statistic.
               > 'bgsadf'  : The backward generalized sup ADF
                  statistic sequence.
               > 'sadf'     : The sup ADF statistic.

- 'start'     : The start date of the recursive ADF testing.

- 'draws'     : Number of draws to be used to construct the critical
               values and the p-values when the 'method' input is
               not set to 'standard'.

```

Output:

```

> When 'method' == 'standard':
- results : A struct with the estimation and test results:
> rhoTTest    : T-statisticts of the estimated coefficient
               of the lag.
> rhoPValue   : P-value of the estimated coefficient of the

```

lag. Compared to the critical values for the Dickey-Fuller Test.

```

> rhoCritValue : The rhoTTTest critical values for significance level 0.1, 0.05 and 0.01. A 1 x 3 double. If rhoTTTest > these critical values you can reject the null hypothesis of a unit root at the given significance level.

> Zdf : Dickey-Fuller rho test statistics.

> ZdfPValue : P-value of the Dickey-Fuller rho test statistics. Compared to the critical values for the Dickey-Fuller Test.

> ZdfCritValue : The Zdf critical values for significance level 0.1, 0.05 and 0.01. A 1 x 3 double. If Zdf > these critical values you can reject the null hypothesis of a unit root at the given significance level.

> When 'method' == 'standard':

> test : The wanted test statistics. May be a vector depending on the 'method' input.

> critValue : The critical values of the test for significance level 0.1, 0.05 and 0.01. A N x 3 double.. N may be > 1 depending on the 'method' input.

> pValue : The p-value(s) of the test statistics. May be a vector depending on the 'method' input.

> In both cases:

- ouput : The estimated equation as a struct of size 1 x numberOfVars. Each element consist of two fields; results and options. See the output of nb_olsEstimator.estimate for more on these output.

```

See also:

[nb_unitRootTest](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_archTest**

```
[stat,pval] = nb_archTest(residual, lags)
```

Description:

Arch test of residuals from a regression.

Input:

```
- residual : A nobs x neq double with the residuals.  
- lags      : The number of lags of the test.
```

Output:

```
- stat      : The test-statistics.  
- pval     : The p-value.
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_autocorr

```
acf = nb_autocorr(y, lags)  
[acf, lBound, uBound] = nb_autocorr(y, lags, errorBound, alpha)
```

Description:

Calculate the autocorrelation of a stationary time series.

Input:

```
- y          : A nobs x nvar double matrix with the time-series.  
- lags      : The number of lags to include  
- errorBound : A string. The data must be stationary. Either:  
               > 'asymptotic'      : Using asymptotically derived  
                           formulas.  
               > 'paramBootstrap'   : Calculated by estimating the  
                           data generating process. And  
                           use an ARIMA model to  
                           simulate artifical  
                           time-series. Then the error  
                           bands are constructed by  
                           the wanted percentile.  
               > 'blockBootstrap'   : Calculated by blocking the  
                           observed series and using  
                           these blocks to simulate  
                           artificial time-series. Then  
                           the error bands are  
                           constructed by the wanted  
                           percentile. See nb_blockBootstrap.  
                           The method use is 'random'.  
               > 'copulaBootstrap'  : The observed series is boostraped  
                           based on a copula approach. See  
                           Paulsen (2017).  
- alpha     : Significance level. As a scalar. Default is 0.05.
```

Optional inputs:

- 'constant' : 1 to include constant in the ARIMA model, otherwise 0. Default is 1.
- 'maxAR' : As an integer. See the nb_arima function. Default is 3.
- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- acf : Autocorrelations function. As a lags x nvar double.
- lBound : Lower bound of the error bound of the autocorrelation function. As a lags x nvar double.
- uBound : Upper bound of the error bound of the autocorrelation function. As a lags x nvar double.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_autocorrMat**

```
acf = nb_autocorrMat(y, lags)
acf = nb_autocorrMat(y, lags, demean)
```

Description:

Calculate the autocorrelation matrix of a set of series.

Input:

- y : A nobs x nvar x nPage double.
- lags : A non-negative integer.
- demean : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- acf : A nvar x nvar x lags + 1 x nPage double. The first page being the correlation matrix of the variables.

See also:

[nb_autocov](#), [nb_autocorr](#), [nb_autocovMat](#)

Written by Kenneth Åtterhagen Paulsen

◆ nb_autocorrMat2

```
acf = nb_autocorrMat2(y, lags)
acf = nb_autocorrMat2(y, lags, demean, shrink)
```

Description:

Calculate the autocorrelation matrix of a set of series.

Input:

- y : A nobs x nvar x nPage double.
- lags : A non-negative integer.
- demean : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.
- shrink : Set to true to use the automatic shrinkage parameter of Kwan (2011). false is default

Can also be set to a number bewteen 0 and 1. For more see the lambda input of the nb_covShrinkDiag functin.

Output:

- acf : A nvar x nvar x lags + 1 x nPage double. The first page being the correlation matrix of the variables.

See also:

[nb_autocov](#), [nb_autocorr](#), [nb_autocovMat2](#), [nb_covShrinkDiag](#)

Written by Kenneth Åtterhagen Paulsen

◆ nb_autocorrTest

```
[stat,pval] = nb_autocorrTest(residual, lags)
```

Description:

Test for autocorrelation of residuals from a regression.

Input:

- residual : A nobs x neq double with the residuals.
- lags : The number of lags of the test.

Output:

- stat : The test-statistics.
- pval : The p-value.

Written by Kenneth Sølnerhagen Paulsen

◆ nb_autocov

```
acf = nb_autocov(y, lags)
[acf, lBound, uBound] = nb_autocov(y, lags, errorBound, alpha)
```

Description:

Calculate the autocovariance of a stationary time series.

Input:

- y : A nobs x nvar double matrix with the time-series.
- lags : The number of lags to include
- errorBound : A string. The data must be stationary. Either:
 - > 'asymptotic' : Using asymptotically derived formulas.
 - > 'paramBootstrap' : Calculated by estimating the data generating process. And use an ARIMA model to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
 - > 'blockBootstrap' : Calculated by blocking the observed series and using these blocks to simulate artificial time-series. Then the error bands are constructed by the wanted percentile. See nb_blockBootstrap. The method use is 'random'.
- alpha : Significance level. As a scalar. Default is 0.05.

Optional inputs:

- 'constant' : 1 to include constant in the ARIMA model, otherwise 0. Default is 1.
- 'maxAR' : As an integer. See the nb_arima function. Default is 3.

- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- acf : Autocovariance function. As a lags x nvar double.
- lBound : Lower bound of the error bound of the autocovariance function. As a lags x nvar double.
- uBound : Upper bound of the error bound of the autocovariance function. As a lags x nvar double.

Written by Kenneth Sølnerhagen Paulsen

◆ nb_autocovMat

```
acf = nb_autocovMat(y, lags)
acf = nb_autocovMat(y, lags, demean)
```

Description:

Calculate the autocovariance matrix of a set of series.

Input:

- y : A nobs x nvar x nPage double.
- lags : A non-negative integer.
- demean : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- acf : A nvar x nvar x lags + 1 x nPage double. The first page being the covariance matrix of the variables.

See also:

[nb_autocov](#), [nb_autocorr](#), [nb_autocorrMat](#)

Written by Kenneth Sølnerhagen Paulsen

◆ nb_autocovMat2

```
acf = nb_autocovMat2(y, lags)
acf = nb_autocovMat2(y, lags, demean, shrink)
```

Description:

Calculate the autocovariance matrix of a set of series.

Input:

- y : A nobs x nvar x nPage double.
- lags : A non-negative integer.
- demean : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.
- shrink : Set to true to use the automatic shrinkage parameter of Kwan (2011). false is default

Can also be set to a number bewteen 0 and 1. For more see the lambda input of the nb_covShrinkDiag functin.

Output:

- acf : A nvar x nvar * (lags + 1) x nPage double. Variables are looped fast, lags slow.

See also:

[nb_autocorrMat2](#), [nb_autocovMat](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_automaticModelSelection**

```
results = nb_automaticModelSelection(y, X, varargin)
```

Description:

Automatic model selection for single-equation models.

References:

- G. Sucarrat and A. Escrivano (2011): "Automated Model Selection in Finance: General-to-Specific Modelling of the Mean and Volatility Specifications"
- D. F. Hendry and H.-M. Krolzig (2005): "The Properties of Automatic Gets Modelling." Economic Journal 115, C32-C61.
- D. F. Hendry and H.-M. Krolzig (2001): "Automatic Econometric Model Selection using PcGets". London: Timberlake Consultants Press.

Input:

- y : A nobs x neqs double. (Dependent variables)
- X : A nobs x nvar double. Regressors.

Optional input:

- 'alpha' : Significance level of the arch, normality and autocorrelation test statistics used for validating models.
- 'constant' : 1 if you want to include constant, else 0. Default is 1.
- 'lags' : A double vector with the lags to test out. E.g. 1:10 or [1,3,4,5-6]
- 'criterion' : Criterion to use to select models from the sample of valid models.
- 'fixed' : The regressors to force to include in the model, and not append, lags.
- 'start' : The index to start the estimation from. Be aware that if you set this to 1 and maxLagLength > 0, then the data will include nan values, so startInd should be at least 1 + max(lags).

Output:

- results : A struct with the model selection results

Original code by Junior Maih (autmatic_model_selection method of the ts class of the RISE toolbox.). Modified by Kenneth Sæterhagen Paulsen

◆ nb_blockBootstrap

```
artificial = nb_blockBootstrap(y, draws, method, blockLength)
```

Description:

Block bootstrap time-series to generate artificial draws from the assumed data generating processes.

The time-series are assumed stationary.

Input:

- y : A nobs x nvar x npage double.
- draws : Number of draws to be made.
- method : The method to use. Either:

```

> 'overlapping'      : Overlapping block bootstrap
> 'nonoverlapping'  : Non-overlapping block bootstrap
> 'random'          : Random length overlapping block
                      bootstrap. Can in some case avoid
                      the bootstrapped sample to be
                      non-stationary.

>

- blockLength : Length of each blocks. Only for the method 'overlapping'
   and 'nonoverlapping'. Default is nobs^(1/3).

```

Output:

- artificial : A nobs x nvar x npage x draws double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_bootstrap

E = nb_bootstrap(resid,replic,method)

Description:

Bootstrap residuals

Input:

```

- resid  : A nobs x nEq double

- replic : The number of bootstrapped residuals. As an integer

- method :

  > 'bootstrap'           : Classical bootstrap by drawing a sample
                             with replacement.

  > 'wildBootstrap'       : Create artificial data by wild
                             bootstrap.

  > 'blockBootstrap'      : Create artificial data by
                             non-overlapping block bootstrap.

  > 'mBlockBootstrap'     : Create artificial data by
                             overlapping block bootstrap.

  > 'rBlockBootstrap'     : Create artificial data by
                             overlapping random block length
                             bootstrap.

  > 'dependentWildBootstrap' : Create artificial data by dependent
                             wild bootstrap.

```

```
> 'wildBlockBootstrap'      : Create artificial data by wild  
                           overlapping random block length  
                           bootstrap.
```

Output:

- E : A nEq x replic x nobs double

Written by Kenneth Sæterhagen Paulsen

◆ **nb_boxPierceQTest**

```
boxQTest = nb_boxPierceQTest(residual,nlag)
```

Description:

Box-Pierce Q-test

Inspired by the vare function of LeSage.

Input:

- residual : A double vector of residuals from an ols estimation.
- nlag : The number of lags. Default is 1. As a scalar.

Output:

- boxQTest : A scalar with the result of the Box-Pierce Q-test.

Written by Kenneth S. Paulsen

◆ **nb_breuschGodfreyTest**

```
[stat,pval] = nb_breuschGodfreyTest(residual,X,lags)
```

Description:

Test for autocorrelation of residuals from a regression using the Breusch-Godfrey test.

Caution: It will test each equation individually.

Input:

- residual : A nobs x neq double with the residuals.
- X : The models regressors, as a nobs x nxvar double. Should not include the constant!
- lags : Number of lags of the residual in the auxiliary regression.

Output:

- stat : The test-statistics.
- pval : The p-value.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_breuschPaganTest**

```
[stat,pval] = nb_breuschPaganTest(residual,X)
```

Description:

Test for heteroscedasticity of residuals from a regression using the Breusch-Pagan test.

This test is corrected for possible non-normality in the residuals based on Koenker(1981) and Koenker and Bassett (1982).

Caution: It will test each equation individually.

Input:

- residual : A nobs x neq double with the residuals.
- X : The models regressors, as a nobs x nxvar double. The constant should not be included.

Output:

- stat : The test-statistics.
- pval : The p-value.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_calcRoots**

```
[realD,imagD,modulus] = nb_calcRoots(model)
```

Description:

Calculates the roots of the companion from of the model.

If the modulus is larger than 1 for any of the roots the model is not stable.

Input:

- A : The transition matrix of the model.

Output:

- realD : The real part of the roots.
- imagD : The imaginary part of the roots.
- modulus : The modulus of the roots.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_cfsTest**

```
test = nb_cfsTest(X,varargin)
```

Description:

This function implements the procedure suggested by Barigozzi and Trapani (2018) to determine the dimension of the common factor space in a large, possibly non-stationary, dataset.

References:

Barigozzi and Trapani (2018), "Determining the dimension of factor structures in non-stationary large datasets", Discussion Papers 18/01, University of Nottingham, Granger Centre for Time Series Econometrics.

Trapani (2018), "A Randomized Sequential Procedure to Determine the Number of Factors", Journal of American Statistical Associations, vol. 113, no. 523, 1341-1349.

Input:

- X : A nobs x nvar double.

Optional input:

- 'alpha' : Significance level of the test. Default is 0.05. Must be in (0,1).
- 'k' : Select the value of the k parameter of the paper. Either '1', 'p' or 'p+1'. Default is '1'.
- 'u' : Select the bound during randomization procedure. Default is sqrt(2). The actual limits will be +- this number.

Output:

```
- test : A struct with fields:  
  
    > 'r'      : Estimated total number of factors.  
  
    > 'r1'     : 1 if the data is estimated to have factors with  
                  linear trends, otherwise 0.  
  
    > 'r2'     : Estimated number of zero-mean I(1) factors;  
                  r2 = rStar - r1.  
  
    > 'r3'     : Estimated number of I(0) factors; r3 = r - rStar.  
  
    > 'rStar'   : Estimated number of non-stationary common factors.
```

Examples:**See also:**

[nb_pca](#), [nb_trapaniNFactorTest](#)

Written by Kenneth Åtterhagen Paulsen

◆ **nb_chowTest**

```
results = nb_chowTest(y,X,constant,timeTrend,breakPoint)
```

Description:

Test for breaks in the estimated coefficients. Null is no breaks.

Input:

- y : A double matrix of size nobs x 1 of the dependent variable of the regression.
- x : A double matrix of size nobs x nxvar of the right hand side variables of the equation of the regression.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- breakpoint : Index of breakpoint. As a numeric scalar.

Output:

```

- results : A struct with the fields:

    > chowTest : Chow test statistic for structural
      break.

    > chowProb : P-value of test statistic.  $F(k, T-2k)$ 
      distributed.

```

See also
nb_ols

Written by Kenneth S. Paulsen

◆ nb_copulaBootstrap

```
artificial = nb_copulaBootstrap(y, draws, method, lags)
```

Description:

Copula bootstrap time-series to generate artificial draws from the assumed data generating processes.

The time-series are assumed stationary.

Input:

```

- y      : A nobs x nvar x npage double.

- draws  : Number of draws to be made.

- method : The method to use. Either:

    > 'autocorr'       : Univariate, autocorrelation robust.

- lags   : Number of lags of the autocorrelation of the series to use.
      Default is min(ceil(nobs^(1/3))), 5.

```

Output:

```
- artificial : A nobs x nvar x npage x draws double.
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_cucconiTest

```
[CT,pValue] = nb_cucconiTest(x,y)
```

Description:

Implements the Cucconi test for equal location and scale of two random samples coming from two different distributions. The H0 is that the distributions are equal. So a p-value < alpha means that you can reject that the distributions are equal.

Input:

- x : A n x 1 double. n > 6.
- y : A m x 1 double. m > 6.

Output:

- CT : The Cucconi test statistics.
- pValue : The p-value of the test.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_durbinWatson**

```
dwtest = nb_durbinWatson(residual)
dwtest = nb_durbinWatson(y,x,beta)
```

Description:

Durbin-Watson Statistic.

Input:

- y : The dependent variable as a double or the residual if only one input is provided. Must be a nobs x neqs double.
- x : The right hand side variables of the regression. Must be a nobs x neqs double.
- beta : Estimated coefficients. As a ncoeff x neqs double.

Output:

- dwtest : Durbin-Watson Statistic. A 1 x neqs double.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_durbinWuHausman**

```
[test,pval] = nb_durbinWuHausman(y,x,z,inst,constant,timeTrend)
```

Description:

Durbin-Wu-Hausman test for regressors exogeneity.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the exogenous variables of all equations of the regression.
- z : A double matrix of size nobs x nzvar of the endogenous variables of all equations of the regression.
- inst : A 1 x nzvar cell of the instruments for each endogenous variable. Each element must be a double with size nobs x nzivar. nzivar must be greater than 0.
- constant : If a constant is wanted in the estimation.
- timeTrend : If a linear time trend is wanted in the estimation.

Output:

- test : Durbin-Wu-Hausman statistic. A 1 x 1 double.
- pval : P-value of the test statistic. The test statistic is distributed as F(nzvar,nobs - nzvar - nxvar).

See also:

[nb_tsls](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_egcoint**

```
results      = nb_egcoint(y)
[results,model] = nb_egcoint(y,varargin)
```

Description:

Engle-Granger cointegration test. Using Dickey-Fuller test for unit root of the residual from the cointegration equation
 $y_1 = X \cdot a + Y_2 \cdot b + e$.

All the variables in the input y will be included in the cointegration test equation.

Input:

- y : An nb_ts object.

Optional input:

```

- 'dependent'      : The variable which are decleared as the
                      left-hand side variable of the cointegration
                      test equation. As a string.

- 'test'           : Either:

> 'adf' : Dickey-Fuller test.

- 'nLags'          : Sets the number of lags of the estimated
                      residual model to base the test upon. Default
                      is 0. ('adf')

- 'maxLags'        : Sets the maximum number of lags used by the
                      lag length criteria. Default is 10. ('adf')

- 'lagLengthCrit' : Lag length selection criterion ('adf').
                     Either:

> 'aic'   : Akaike information criterion

> 'aicc'  : Corrected Akaike information
                     criterion

> 'maic'  : Modified Akaike information
                     criterion

> 'sic'   : Schwarz information criterion

> 'msic'  : Modified Schwarz information
                     criterion

> 'hqc'   : Hannan-Quinn information criterion

> 'mhqc'  : Modified Hannan-Quinn information
                     criterion

> ''      : Manually set lag length.

- 'model'          : y1 = X*a + Y2*b + e ('c' is default):

> 'no'   : No constant or trend in X.

> 'c'    : Constant but no trend in X.

> 'ct'   : Constant and linear trend in X.

> 'ctt'  : Constant, linear trend, and
                     quadratic trend in X.

```

Output:

```

- results : A struct with the estimation and test results:

> rhoTTest       : T-statisticts of the estimated coefficient
                      of the lag. ('adf')

> rhoPValue      : P-value of the estimated coefficient of the
                      lag. Compared to the critical values for the
                      (spurious regression adjusted) Dickey-Fuller

```

```

Test. ('adf')

> rhoCritValue : The rhoTTTest critical values for significance
level 0.1, 0.05 and 0.01. A 1 x 3 double.
If rhoTTTest > these critical values you can
reject the null hypothesis of a unit root (of
the residual) at the given significance
level. ('adf')

> cointVec : The cointegrated vector. Will include the
a as well. I.e. [a,b].

- model : The estimated equation as 1 x 2 struct. See the output
of the nb_olsEstimator function for more on these.

> model(1) : OLS estimation of the cointegration eq.

> model(2) : OLS estimation of the residual unit root
test eq.

```

See also:

[nb_cointTest](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_fTest**

```
[fTest,fProb] = nb_fTest(residual1,residual2,nCoeff1,nCoeff2)
```

Description:

Calculates the F-statistics for beta = 0 restrictions.

```
RSS1 = residual1'*residual1;
RSS2 = residual2'*residual2;
fTest = ((RSS1 - RSS2)/(nCoeff2 - nCoeff1))/(RSS2/(T - nCoeff2));
```

Input:

- residual1 : Residuals from the restricted model.
- residual2 : Residuals from the unrestricted model.
- nCoeff1 : Number of estimated coefficients of the restricted model.
- nCoeff2 : Number of estimated coefficients of the unrestricted model.

Output:

- fTest : F-statistic

- fProb : F-test statistic p-value.

Written by Kenneth Sæterhagen Paulsen

◆ nb_grangerTest

```
[F,FPValue] = nb_grangerTest(x,y,alpha,max_lag)
```

Description:

Granger Causality test. Does Y Granger Cause X?

References:

[1] Granger, C.W.J., 1969. "Investigating causal relations by econometric models and cross-spectral methods". *Econometrica* 37 (3), 424438.

Input:

- x : A nobs x 1 double.
- y : A nobs x 1 double.
- alpha : The significance level of the test.
- max_lag : The maximum number of lags to be considered.
- criterion : Lag length selection criterion. See the nb_lagLengthSelection function for more on this input. Default is 'aic'.

Output:

- F : The value of the F-statistic
- FPValue : The p-value of the F-statistic.

Written by Kenneth Sæterhagen Paulsen

◆ nb_hornParallelAnalysis

```
[sMEV,ciEV,cumE,resL2,nBasis] = nb_hornParallelAnalysis(data,K)
```

Description:

This function simulates a distribution of eigenvalues by resampling a set of random variables of the real data size, and compares the eigenvalues of the real data and the distribution of eigenvalues from simulation; then the number of retained basis factors is decided by keeping those who are bigger than 95% of simulated distribution of eigenvalues. This is to implement the parallel analysis approach proposed by Horn (1965) and developed by Ledesma et al. (2007).

Input:

- data : Standardized real data (subtract mean and divide by std for each variable);
- K : Number of resampling.

Output:

- sMEV : Mean eigenvalues from simulation.
- ciEV : Confidence intervals of eigenvalues from simulation.
- cumE : cumulative energy content / total energy content
- resL2 : L2 norm of residual for each eigenvalue of real data;
- nBasis : Number of retained basis factors.

Written by Lanya T. Cai on June 22 2016.

Contact lanyavikins[at]gmail.com if there's any question.

Edited by Kenneth SÃ¶terhagen Paulsen

- Corrected documentation to fit NB toolbox

◆ nb_infoCriterion

```
infoCrit = nb_infoCriterion(type, loglogLikelihood, T, numCoeff)
```

Description:

Calculates the information criterion for a model given the logLikelihood, sample size and number of coefficients.

All input but the first can be given as vectors (of same size).

Input:

- type : Either:
 - > 'aic' : Akaike information criterion.
 - > 'aicc' : Corrected Akaike information criterion.
 - > 'maic' : Modified Akaike information criterion.
 - > 'sic' : Schwarz information criterion.
 - > 'msic' : Modified Schwarz information criterion.
 - > 'hqc' : Hannan and Quinn information criterion.
 - > 'mhqc' : Modified Hannan and Quinn information criterion.
- loglogLikelihood : The models log logLikelihood. As a 1 x neq

```

double.

- T          : Size of estimation sample. As a scalar.

- numCoeff    : The number of estimated coefficients of the
               model. A 1 x neq double.

- kappa       : Modifiying element for the 'maic', 'msic'
               and 'mhqc' criterion. A 1 x neq double.

```

Output:

```
- infoCrit : The calculated information criterion. A 1 x neq
            double.
```

Written by Kenneth Sæterhagen Paulsen

◆ **nb_jcoint**

```
results      = nb_jcoint(y)
[results,models] = nb_jcoint(y,varargin)
```

Description:

Johansen cointegration test.

All the variables in the input y vil be included in the cointegration test equation.

There are still some unfinished business here!!!!

Input:

```
- y          : An nb_ts object or a nobs x nVars double.
```

Optional input:

```
- 'model'      : (1-L)y(t) = A*B'*y(t-1) + D*x
                  + B1*(1-L)y(t-1) + ... + Bq*(1-L)y(t-q)
                  + e(t)

                  Form of A*B'*y(t-1) + D*x

> 'H2'   : A*B'*y(t-1). There are no intercepts
            or trends in the cointegrating
            relations and there are no trends
            in the data. This model is only
            appropriate if all series have zero
            mean. Default.

> 'H1*' : A*(B'*y(t-1) + c0). There are
            intercepts in the cointegrating
            relations and there are no trends
            in the data. This model is
```

appropriate for nontrending data
with nonzero mean.

```

> 'H1'   : A*(B'*y(t-1) + c0) + c1. There are
           intercepts in the cointegrating
           relations and there are linear
           trends in the data. This is a model
           of "deterministic cointegration,"
           where the cointegrating relations
           eliminate both stochastic and
           deterministic trends in the data.
           This is the default value.

> 'H*'   : A*(B'*y(t-1) + c0 + d0*t) + c1.
           There are intercepts and linear
           trends in the cointegrating
           relations and there are linear
           trends in the data. This is a model
           of "stochastic cointegration,"
           where the cointegrating relations
           eliminate stochastic but not
           deterministic trends in the data.

> 'H'    : A*(B'*y(t-1) + c0 + d0*t) + c1 + d1*t.
           There are intercepts and linear
           trends in the cointegrating
           relations and there are quadratic
           trends in the data. Unless quadratic
           trends are actually present in the
           data, this model may produce good
           in-sample fits but poor
           out-of-sample forecasts.

'hypo'      : String. Either ('trace' is default):

> 'trace'  : The alternative hypothesis is
           H(num). Statistics are:
           -T*[log(1-lambda(h+1)) + ... +
           log(1-lambda(num))]

> 'maxeig'. The alternative hypothesis is
           H(h+1).

           -T*log(1-lambda(h+1))

- 'nLags'   : Sets the number of lags of the estimated
               model to base the test upon. Default is 0.

- 'maxLags' : Sets the maximum number of lags used by the
               lag length criteria. Default is 10.

- 'lagLengthCrit' : Lag length selection criterion. Either:

> 'aic'    : Akaike information criterion

> 'aicc'   : Corrected Akaike information
               criterion
```

```

> 'maic' : Modified Akaike information
criterion

> 'sic' : Schwarz information criterion

> 'msic' : Modified Schwarz information
criterion

> 'hqc' : Hannan-Quinn information criterion

> 'mhqc' : Modified Hannan-Quinn information
criterion

> ''      : Manually set lag length. Default

- 'mlresults' : Give 1 if you want to report the maximal
likelihood estimates.

```

Output:

```

- results : A struct with the estimation and test results:

    Test results:

        > lambda    : Eigenvalues. 1 x h double.*

        > eigVec    : The corresponding eigenvectors.

        > testStat   : Test-statistic. 1 x h double.*

        > pValue     : P-values. 1 x h double.*

        > critValue : Critical values. 3 x h double.* 1%
                      5% and 10%.

```

*h = Max number of possible cointegrated relations.

```

- models : The estimated equation as 1 x 2 struct. See the results
output of the nb_olsEstimator.estimate function for more
on this output.

    > models(1) : OLS estimation of the differenced eq.

    > models(2) : OLS estimation of the lagged level eq.

```

See also:

[nb_cointTest](#)

Written by Kenneth Åkerhagen Paulsen

◆ **nb_lagLengthSelection**

```

nlags = nb_lagLengthSelection(y,X,constant,time_trend, ...
                               maxLagLength,criterion,method)

```

Description:

Lag length selection algorithm.

Input:

- constant : 1 if constant should be included in the regression, otherwise 0.
- time_trend : 1 if time trend should be included in the regression, otherwise 0.
- maxLagLength : The maximal number of tested lags. As a double.
- criterion : The criterion to use for the selection.
 - > 'aic' : Akaike information criterion
 - > 'maic' : Modified Akaike information criterion
 - > 'sic' : Schwarz information criterion
 - > 'msic' : Modified Schwarz information criterion
 - > 'hqc' : Hannan-Quinn information criterion
 - > 'mhqc' : Modified Hannan-Quinn information criterion
- method : The estimation method to use.
 - > 'ols' : Ordinary least squares. Assumes that the residual are normally distributed.
 - > 'quantile' : Quantile regression. Assumes that the residual are normally distributed. This may be a bad assumption!
- y : Dependent variable. As an nobs x 1 double.
- X : Exogenous regressors. As an nobs x nvar double.
- fixed : Indicate if an exogenous regressor should have fixed lag length. A 1 x nvar logical.
- startInd : The index to start the estimation from. Be aware that if you set this to 1 and maxLagLength > 0, then the data will include nan values, so startInd should be at least 1 + maxLagLength.

Output:

- nlags : Number of (extra) lags selected for the provided model.
- y : Dependent variable(s). Shrunked to the new sample.

- X : Selected regressors. (Without constant and time-trend). Shrunked to the new sample.

Written by Kenneth Sæterhagen Paulsen

◆ nb_ljungBoxTest

```
[stat,pval] = nb_ljungBoxTest(residual,k)
```

Description:

Test for autocorrelation of residuals from a regression using the Ljung-Box test. Only for a VAR model.

Input:

- residual : A nobs x neq double with the residuals.
- k : Number of lags of the VAR, as an integer.

Output:

- stat : The test-statistics.
- pval : The p-value.

Written by Kenneth Sæterhagen Paulsen

◆ nb_normalityTest

```
[stat,pval] = nb_normalityTest(residual)
[stat,pval] = nb_normalityTest(residual,k)
```

Description:

This is the Jarque-Bera test, which test for normality. Null hypothesis of the test is that the series y is normal.

Input:

- residual : A nobs x neq double.
- k : Number of estimated parameters of the model.

Output:

- stat : The test-statistic. As a 1 x neq double.
- pval : The p-values. As a 1 x neq double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_olsConfInt

```
confInt = nb_olsConfInt(beta, stdBeta, nobs)
```

Description:

Confidence intervals of estimated parameters.

Input:

- beta : A nvar x 1 double with the estimated parameters.
- stdBeta : A nvar x 1 double with the std of the estimated parameters.
- nobs : Number of observation of the estimation. As an integer.
- alpha : The critical value. Default is 0.05.

Output:

- confInt : Confidence intervals. A nvar x 2 matrix. With lower and upper limit.

Examples:

See also:

Written by Kenneth Sætherhagen Paulsen

◆ nb_olsLikelihood

```
logLikelihood = nb_olsLikelihood(residual)
logLikelihood = nb_olsLikelihood(residual, method, numCoeff)
```

Description:

Calculate the log likelihood of an ols regression based on the residuals.

Input:

- residual : The residuals from the ols regression. As a nobs x neqs double. The equations must be unrelated or have the same regressors.
- method : Either 'full' (full likelihood of the system) or 'single' (individually likelihood calculations).
 - 'single' : Uses the equation (24.9) in EViews 6 User's Guide 2
 - 'full' : Uses the equation (34.4) in EViews 6 User's Guide 2

Output:

- logLikelihood : The log likelihood of the ols regression.
As a 1 x neqs double.

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_parcorr**

```
pacf = nb_parcorr(y, lags)
[pacf, lBound, uBound] = nb_parcorr(y, lags, errorBound, alpha)
```

Description:

Calculate the partial autocorrelation of a stationary time series.

Input:

- y : A nobs x nvar double matrix with the time-series.
- lags : The number of lags to include
- errorBound : A string. The data must be stationary. Either:
 - > 'asymptotic' : Using asymptotically derived formulas.
 - > 'paramBootstrap' : Calculated by estimating the data generating process. And use an ARIMA model to simulate artificial time-series. Then the error bands are constructed by the wanted percentile.
 - > 'blockBootstrap' : Calculated by blocking the observed series and using these blocks to simulate artificial time-series. Then the error bands are constructed by the wanted percentile. See nb_blockBootstrap. The method use is 'random'.
- alpha : Significance level. As a scalar. Default is 0.05.

Optional inputs:

- 'constant' : 1 to include constant in the ARIMA model, otherwise 0. Default is 1.
- 'maxAR' : As an integer. See the nb_arima function. Default is 3.

- 'maxMA' : As an integer. See the nb_arima function. Default is 3.
- 'criterion' : As a string. See the nb_arima function. Default is 'aicc'.
- 'method' : As a string. See the nb_arima function. Default is 'hr'.

Output:

- pacf : Partial autocorrelations function. As a lags x nvar double.
- lBound : Lower bound of the error bound of the partial autocorrelation function. As a lags x nvar double.
- uBound : Upper bound of the error bound of the partial autocorrelation function. As a lags x nvar double.

Written by Kenneth Å!terhagen Paulsen

◆ **nb_phillipsPerron**

```
results = nb_phillipsPerron(y)
[results,model] = nb_phillipsPerron(y,varargin)
```

Description:

Phillips-Perron tests for unit roots.

Input:

- y : An nb_ts object.

Optional input:

- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use the Newey-West selection criterion.
- 'bandWidthCrit' : Band width selection criterion. Either:
 - > 'nw' : Newey-West selection method.
Default. (Newey West, 1994)
 - > 'a' : Andrews selection method.
(Andrews, 1991)
- 'freqZeroSpectrumEstimator' :
 - Either:
 - > 'bartlett' : Bartlett kernel function.

```

Default.

> 'parzen'      : Parzen kernel function.

> 'quadratic'  : Quadratic Spectral kernel
                  function.

- 'model'        : Either:

> 'ar'   :  $y(t) = \rho y(t-1) + e(t)$  (Default)

> 'ard'  :  $y(t) = \alpha + \rho y(t-1) + e(t)$ 

> 'ts'   :  $y(t) = \alpha + \delta t + \rho y(t-1) + e(t)$ 

```

Output:

- results : A struct with the estimation and test results
(See page 514 in Hamilton):
 - > Zt : Zt test statistics.
 - > ZtPValue : P-value of the Zt test statistics.
 - > ZtCritValue : The Zt critical values for significance level 0.1, 0.05 and 0.01. A 1 x 3 double.
If Zdf > these critical values you can reject the null hypothesis of a unit root at the given significance level.
- model : The estimated equation as a struct with fields results and options. See the nb_olsEstimator.estimate function for more.

See also:

[nb_unitRootTest](#)

Written by Kenneth Åtterhagen Paulsen

◆ nb_quantileWaldTest

```
[wTest,wProb] = nb_quantileWaldTest(A,b,X,beta,residual,q)
```

Description:

Calculates the Wald-statistics for a general linear restriction.

$A * \beta = c$

This will be the null hypotheses, and we can reject it if wProb < alpha, where alpha is the choosen significance level.

Input:

- A : A nrestrictions x ncoeff*nq double with the linear restrictions of the estimated coefficients (beta).
- c : A nrestrictions x 1 double with what the linear restrictions should equal.
- X : The right hand side regressors. As a nobs x ncoeff matrix.
- beta : A ncoeff*nq x 1 double with the estimated coefficients. We allow for beta to include estimated coefficients of different quantiles q (nq>1). We order the coefficients [coeff(q(1));coeff(q(2));...], i.e. in the order of the input q.
- residual : A nobs x nq double with the residual from the regression. NOTE: Not in use at the moment! If we allow for departure IID and the normality approximation we need to use an empirical estimate of the sparsity function based on these residuals!
- q : A 1 x nq double with the quantiles. Must be in [0,1].

Output:

- wTest : Wald-statistic
- wProb : Wald-test statistic p-value.

Written by Kenneth Sæterhagen Paulsen

◆ nb_rSquared

```
[rSquared,adjRSquared] = nb_rSquared(y,x,beta)
```

Description:

Calculate R-Squared and/or adjusted R-Squared of (an) estimated equation(s).

Inspired by the vare function of LeSage.

Input:

- y : A double vector with the dependent variable of the regression. A double matrix nobs x neqs.
- residual : The residual of the regression. A double vector nobs x neqs.
- numCoeff : The number of estimated parameters of the model. As 1 x neqs double.

Output:

```
- rSquared      : As 1 x neqs double.  
- adjRSquared : As 1 x neqs double.
```

Written by Kenneth S. Paulsen

◆ nb_restrictedFTest

```
[fTest,fProb] = nb_restrictedFTest(A,b,X,beta,residual)
```

Description:

Calculates the F-statistics for a general linear restriction.

A*beta = c

This will be the null hypotheses, and we can reject it if fProb < alpha, where alpha is the choosen significance level.

Input:

- A : A nrestrictions x ncoeff double with the linear restrictions of the estimated coefficients (beta).
- c : A nrestrictions x 1 double with what the linear restrictions should equal.
- X : The right hand side regressors. As a nobs x ncoeff matrix.
- beta : A ncoeff x 1 double with the estimated coefficients
- residual : A nobs x 1 double with the residual from the regression.

Output:

- fTest : F-statistic
- fProb : F-test statistic p-value.

Written by Kenneth Sæterhagen Paulsen

◆ nb_sarganHansen

```
[test,pval] = nb_sarganHansen(residual,x,z,inst,overident,constant,...  
                                timeTrend)
```

Description:

Sargan-Hansen test for instruments validity. Only for overidentified models.

For more see the documentation of DAG.

Input:

- residual : A double matrix of size nobs x neq of the residual from the overidentified regression(s).
- x : A double matrix of size nobs x nxvar of the exogenous variables of all equations of the regression.
- inst : A nobs x nivar double of the instruments for all the endogenous variables.
- overident : The number of overidentifying restrictions. I.e. the number of instruments which are included in the "equations" of the endogenous variables, but not in the estimated equation.
- constant : If a constant is wanted in the estimation.
- timeTrend : If a linear time trend is wanted in the estimation.

Output:

- test : Sargan-Hansen statistic. A 1 x 1 double.
- pval : P-value of the test statistic. The test statistic is distributed as F(nzvar,nobs - nzvar - nxvar).

Written by Kenneth Sæterhagen Paulsen

◆ nb_shockDecomp

```
decomp = nb_shockDecomp(model,options,results,startInd,endInd,inputs)
```

Description:

Shock decomposition of a model on the companion form as below.

Caution : For models with unobservables the filter method must be run first.

Caution : This method does not allow parameter uncertainty for filtered models.

Input:

- model : A struct storing the companion form of the model:

$$y_t = A*y_{t-1} + B*x_t + C*e_t$$

Fields:

> A : See the equation above. A nendo x nendo double.

> B : See the equation above. A nendo x nexo

```

double.

> C      : See the equation above. A nendo x nresidual
double.

> endo   : A cellstr with the decleared endogenous
variables.

> exo    : A cellstr with the decleared exogenous
variables.

> res    : A cellstr with the decleared
residuals/shocks.

> vcv    : Variance/covariance matrix of the
residuals/shocks.

- options   : Estimation options. Output from the estimator functions.
E.g. nb_olsEstimator.estimate.

- results   : Estimation results. Output from the estimator functions.
E.g. nb_olsEstimator.estimate.

- startInd : Start index of the shock decomposition.

- endInd   : End index of the shock decomposition.

- inputs    : See nb_model_generic.shock_decomposition.

```

Output:

```

- decomp   : If perc is empty; A nObs x [nShock (1:end-1) + initial
conditions (end-1) + steady-state (end)] x nVar, otherwise;
A nObs x [nShock (1:end-1) + initial conditions (end-1) +
steady-state (end)] x nVar x nPerc + 1, where the closest
model to the median is located at decomp(:,:, :, end)

Caution : Initial conditions are all that is not explained
by the shocks.

Caution : Please have in mind that the percentiles are
the numerically calculated percentiles and will
not sum up the actual series!

```

See also:

[nb_model_generic.shock_decomposition](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_smirnovTest**

[ST,pValue] = nb_smirnovTest(x,y,m,n)

Description:

Implements the Smirnov test for equal distributions of two random samples coming from two different distributions. The H0 is that the distributions are equal. So a p-value < alpha means that you can reject that the distributions are equal.

Caution: It is assumed that the provided CDF are given at the same domain!

Input:

- x : A n x 1 double with the CDF of the first distribution. n > 6.
- y : A m x 1 double with the CDF of the second distribution. m > 6.
- m : Number of observations the estimation of the CDF of the first distribution is based on.
- n : Number of observations the estimation of the CDF of the second distribution is based on.

Output:

- CT : The Smirnov test statistics.
- pValue : The p-value of the test.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_spearmanTest**

```
[tTest,pVal] = nb_spearmanTest(X,beta,resid)
```

Description:

Spearman correlation test for heteroskedacity.

Input:

- X : The right hand side variables of the regression. Must be a nobs x 1 double.
- beta : Estimated coefficients. As a ncoeff x 1 double.
- resid : Residual of regression. Must be a nobs x 1 double

Output:

- tTest : Spearman Statistic. A 1 x 1 double.
- pVal : P value of test.

Written by Kenneth Sæterhagen Paulsen

◆ nb_trapaniNFactorTest

```
r = nb_trapaniNFactorTest(X)
```

Description:

This function implements a test for the number of common components in a large dataset based on the eigenvalues of the matrix $X'X/T$.

References:

Trapani (2018), "A Randomized Sequential Procedure to Determine the Number of Factors", Journal of American Statistical Associations, vol. 113, no. 523, 1341-1349.

Input:

- X : A nobs x nvar double.

Optional input:

- 'alpha' : Significance level of the test. Default is 0.05. Must be in (0,1).
- 'k' : Select the value of the k parameter of the paper. Either '1', 'p' or 'p+1'. Default is '1'.
- 'u' : Select the bound during randomization procedure. Default is $\sqrt{2}$. The actual limits will be \pm this number.

Output:

- 'r' : Estimated total number of factors.

See also:

[nb_pca](#), [nb_cfsTest](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_varFTest

```
[fTest,fProb] = nb_varFTest(y,nlag,x,beta,constant)
```

Description:

Joint F-test

Inspired by the vare function of LeSage.

Input:

```

- y           : A double vector with the dependent variable of the
                regression. A double vector nobs x 1.

- nlag        : The number of lags of the VAR. Default is 1.

                !!! Should be made specific to each endogenous var.

- x           : Dependent or number of inputs:

                - 2 : Taken as the residual of the regression.
                      A double vector nobs - nlag x 1.

                - 3 : Taken as the right hand side matrix of the
                      estimation. A double vector nobs x nxvar.

- beta        : The estimated parameters of the model. As a double
                vector of size nxvar x 1.

```

Output:

```
- boxQTest : A scalar with the result of the Box-Pierce Q-test.
```

Examples:

Written by Kenneth S. Paulsen

◆ **nb_whiteTest**

```
[test,prob] = nb_whiteTest(residual,X)
```

Description:

White (1980) test for heteroscedasticity.

Input:

```
- residual : Residuals from the model. As a nobs x 1 double.

- X         : Regressors from the model. As a nobs x nxvar double. Should
               not include a constant.
```

Output:

```
- test   : Test statistic

- fProb : P-value of test
```

Written by Kenneth Sæterhagen Paulsen

25.15 Distributions

- nb_ast_b
- nb_ast_k
- nb_copula
- nb_copularnd
- nb_covrepair
- nb_drawCov
- nb_invpsi
- nb_lognormal_deriv
- nb_mcmc.NUTS
- nb_mcmc.adaptiveRandomWalkRecTarget
- nb_mcmc.dualAveraging
- nb_mcmc.gelmanRubinRecursive
- nb_mcmc.leapfrog
- nb_mcmc.mhSampler
- nb_mcmc.mvnRandChol
- nb_mcmc.optimset
- nb_mode
- nb_mvncdf
- nb_mvnrndChol
- nb_mvtnrand
- nb_tAbsMoment
- qsimvn
- nb_ast_cstar
- nb_chol
- nb_copulacondrnd
- nb_covShrinkDiag
- nb_distribution
- nb_fStatPValue
- nb_ksdensity
- nb_mci
- nb_mcmc.adaptiveRandomWalk
- nb_mcmc.adaptiveRandomWalkTarget
- nb_mcmc.gelmanRubin
- nb_mcmc.geweke
- nb_mcmc.loadBetaFromOuput
- nb_mcmc.mvnRand
- nb_mcmc.nutSampler
- nb_mcmc.randomWalk
- nb_mvExpNorm
- nb_mvnrnd
- nb_mvtncondrnd
- nb_normal_deriv
- nb_tStatPValue

► nb_mcmc.NUTS ↑

```
[theta, alpha_ave, nfevals, logp, grad] = nb_mcmc.NUTS(f, epsilon, ...
theta0, maxTreeDepth, lb, ub, logp0, grad0)
```

Description:

Carries out one iteration of No-U-Turn-Sampler.

Input:

- f : A function_handle that returns the log probability of the target and its gradient.

- epsilon : A scalar double. Step size of leap-frog integrator.
- theta0 : A nVar x 1 double. Current state of a Markov chain and the initial state for the trajectory of Hamiltonian dynamics.
- maxTreeDepth : An integer with the maximum depth of tree.
- lb : A nVar x 1 double with the lower bounds on the parameters. Give -inf for elements that is not bounded. Give [] if no parameters are bounded.
- ub : A nVar x 1 double with the upper bounds on the parameters. Give inf for elements that is not bounded. Give [] if no parameters are bounded.
- logp0 : A nVar x 1 double. The log probability computed at theta0.
- grad0 : A nVar x 1 double. Gradient of the log probability computed at theta0.

Output:

- theta : Next state of the chain, as a nVar x 1 double.
- alpha_ave : Average acceptance probability of the states proposed at the last doubling step of NUTS. It measures of the accuracy of the numerical approximation of Hamiltonian dynamics and is used to find an optimal step size value epsilon.
- nfevals : The number of log likelihood and gradient evaluations one iteration of NUTS required
- logp : Log probability computed at theta, as a scalar double.
- grad : A nVar x 1 double. The gradient computed at theta.

See also:

[nb_mcmc.dualAveraging](#), [nb_mcmc.leapFrog](#), [nb_mcmc.nutSampler](#)

Written by Matthew D. Hoffman

Edited by Kenneth Sæterhagen Paulsen

- Updated the documentation in line with NB Toolbox.
- Removed the global variable nfevals.
- Removed notifications when maximum tree depth where reached.
- Removed the optional inputs.
- Added the lower and upper bounds inputs.

► **[nb_mcmc.adaptiveRandomWalk](#)** ↑

[betaDraw,output] = nb_mcmc.adaptiveRandomWalk(output)

Description:

Draw one candidate during Metropolis-Hastings algorithm using the random walk assumption.

In this case the covariance matrix is updated based on the kept draws from the draws of the parameters. See Haario et al. (2001).

Input:

- output : See the output of the nb_mcmc.mhSampler function.

Output:

- betaDraw : A nVar x 1 double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mcmc.adaptiveRandomWalkRecTarget ↑**

[betaDraw,output] = nb_mcmc.adaptiveRandomWalkRecTarget(output)

Description:

Draw one candidate during Metropolis-Hastings algorithm using the random walk assumption.

In this case the covariance matrix is updated based on the draws of the parameters. See Haario et al. (2001).

Also the scaling of the covariance matrix is updated for every X simulation. I.e. each time we check the covariance matrix, it is scaled so to make the acceptance ratio equal to the target.

Input:

- output : See the output of the nb_mcmc.mhSampler function.

Output:

- betaDraw : A nVar x 1 double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_mcmc.adaptiveRandomWalkTarget](#)** ↑

```
[betaDraw,output] = nb_mcmc.adaptiveRandomWalkTarget(output)
```

Description:

Draw one candidate during Metropolis-Hastings algorithm using the random walk assumption.

In this case the scaling of the covariance matrix is updated for every X simulation. I.e. each time we check the covariance matrix, it is scaled so to make the acceptance ratio equal to the target.

Input:

- output : See the output of the nb_mcmc.mhSampler function.

Output:

- betaDraw : A nVar x 1 double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_mcmc.dualAveraging](#)** ↑

```
[theta, epsilonbar, epsilon_seq, epsilonbar_seq, logp, grad] = ...
    nb_mcmc.dualAveraging(f, theta0, delta, n_warmup, ...
        maxTreeDepth, lb, ub, n_updates, waitbar)
```

Description:

Adjusts the step-size of NUTS by the dual-averaging (stochastic optimization) algorithm of Hoffman and Gelman (2014).

Input:

- f : A function-handle that returns the log probability of the target and its gradient.
- theta0 : A nVar x 1 double. Initial state for the trajectory of Hamiltonian dynamics.
- delta : A scalar double in the range [0, 1]: the target "acceptance rate" of NUTS.

- n_warmup : A scalar integer with the number of NUTS iterations for the dual-averaging algorithm.
- maxTreeDepth : An integer with the maximum depth of tree.
- lb : A nVar x 1 double with the lower bounds on the parameters. Give -inf for elements that is not bounded. Give [] if no parameters are bounded.
- ub : A nVar x 1 double with the upper bounds on the parameters. Give inf for elements that is not bounded. Give [] if no parameters are bounded.
- n_updates : A scalar integer with the number of periods between updates.
- waitbar : Either a scalar logical or a nb_waitbar object. Default is false.

Output:

- theta : Last state of the chain after the step-size adjustment, which can be used as the initial state for the sampling stage (no need for additional 'burn-in' samples)
- epsilonbar : A scalar double with the step-size corresponding to the target "acceptance rate".
- epsilon_seq : A n_warmup x 1 double with the value of epsilon at each step of the dual averaging algorithm.
- epsilonbar_seq : A n_warmup x 1 double with the value of epsilonbar at each step of the dual averaging algorithm.
- logp : Log probability computed at theta(end), as a scalar double.
- grad : A nVar x 1 double. The gradient computed at theta(end).

See also:

[nb_mcmc.NUTS](#), [nb_mcmc.leapFrog](#), [nb_mcmc.nutSampler](#)

Written by Matthew D. Hoffman

Edited by Kenneth Sæterhagen Paulsen

- Added the waitbar option.
- Change the interpretation of the n_updates input.
- Removed some of the optional inputs.
- Added the maxTreeDepth.
- Added the lower and upper bounds inputs.

► [nb_mcmc.gelmanRubin](#) ↑

```
R = nb_mcmc.gelmanRubin(varargin)
```

Description:

Calculates the Gelman–Rubin diagnostic of multiple M–H chains.

Input:

- varargin : Each input must be $2*n \times q$ double with the M–H draws of a set of q variables/parameters and length $2*n$. The test discard the first n draws.

Output:

- R : A $1 \times q$ test statistics. If any element is $>> 1$ run the M–H for a longer period, as it has not converged to the stationary distribution.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_mcmc.gelmanRubinRecursive](#)** ↑

```
R = nb_mcmc.gelmanRubinRecursive(varargin)
```

Description:

Calculates the recursive Gelman–Rubin diagnostic of multiple M–H chains.

Input:

- varargin : Each input must be $2*n \times q$ double with the M–H draws of a set of q variables/parameters and length $2*n$. The test discard the first n draws.

Output:

- R : A $n \times q$ test statistics. If any element is $>> 1$ run the M–H for a longer period, as it has not converged to the stationary distribution.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► nb_mcmc.geweke ↑

```
[z,p] = nb_mcmc.geweke(beta)
```

Description:

Test for convergence of a M-H chain.

Geweke (1992) test. This test split sample into two parts. The first 10% and the last 50%. If the chain is at stationarity, the means of two samples should be equal. The null hypothesis is that the subsamples has the same mean.

Input:

- beta : A draws x numVar double.

Output:

- z : A 1 x numVar double with the difference in mean statistics.

- p : A 1 x numVar double with the p-values of the test statistics.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► nb_mcmc.leapfrog ↑

```
[thetaprime, rprime, gradprime, logpprime] = ...
    nb_mcmc.leapfrog(theta, r, grad, epsilon, f)
```

Description:

Carries out the leapfrog step of the NUTS algorithm.

Input:

- theta : A nVar x 1 double. Current state of a Markov chain and the initial state for the trajectory of Hamiltonian dynamics.

- r : The current state of the momentum. As a nVar x 1 double.

- grad : A nVar x 1 double. Gradient of the log probability computed at theta.

- epsilon : A scalar double. Step size of leap-frog integrator.

- f : A function_handle that returns the log probability of the target and its gradient.

Output:

- thetaprime : Next state of the chain, as a nVar x 1 double.
- rprime : Next state of the momentum, as a nVar x 1 double.
- gradprime : A nVar x 1 double. Gradient of the log probability computed at thetaprime.
- logpprime : The log probability at thetaprime, as a scalar double.

See also:

[nb_mcmc.NUTS](#), [nb_mcmc.dualAveraging](#)

Written by Matthew D. Hoffman

Edited by Kenneth Sætherhagen Paulsen

- Separated out as an own function.
- Added documentation.
- Added the lower and upper bound on the parameters.

► **[nb_mcmc.loadBetaFromOuput](#)** ↑

```
beta = nb_mcmc.loadBetaFromOuput(output,chain)
```

Description:

Load the sampled draws of the parameters from files.

Input:

- output : The output struct on of the samplers of the nb_mcmc package.
- chain : Sepcify the loaded chain. If empty all chains are loaded.

Output:

- beta : The sampled parameters as a nDraws x nPar x nChains double.

See also:

[nb_mcmc.mhSampler](#), [nb_mcmc.nutSampler](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_mcmc.mhSampler](#)** ↑

```
output = nb_mcmc.mhSampler(objective,beta,sigma,varargin)
```

Description:

Implementation of the Metropolis-Hastings algorithm.

Input:

- objective : A function handle that represents a function that is proportional to the target distribution. Takes one input, and that are the simulated random variables, which has the same type and size as the beta input.
- beta : Initial values of the random variables to draw from. E.g. the mode found under optimization in the case of parameter estimates from a DSGE model. As a $1 \times \text{numVar}$ double.
- sigma : Initial guess on the covariance matrix of the random variables. E.g. the inverted Hessian found during estimation of parameters of a DSGE model. As a $\text{numVar} \times \text{numVar}$ double matrix.

Optional input:

- 'accTarget' : Sets the preferred acceptance ratio when 'adaptive' is set to 'target' or 'recTarget'. Default is 0.5.
 - 'accScale' : The value to re-scale the cholesky factorization of the covariance matrix with when acceptance ratio is less or greater than the target ('accTarget'). I.e. $\exp(\text{accScale} \cdot a)$, where a is incremented by +/- 1 if acceptance ratio is higher/lower than the target.
 - 'adaptive' : Either;
 - > 'recursive' : Calculates the covariance matrix based on the simulated observation from the target function. Uses the approach suggested by Haario et al. (2001). Use the 'weight' input to adjust the weight on the initial covariance matrix.
 - > 'recTarget' : Combining the approaches in 'recursive' and 'target'.
 - > 'target' : Scales the initial covariance matrix with a constant (that changes) to get the wanted acceptance rate. See 'accTarget', 'recursive' is default.
- Caution: Only applies for when the 'qFunction' input is set to 'arw'. For more see the nb_mcmc.adaptiveRandomWalk and nb_mcmc.adaptiveRandomWalkUpdate function.
- 'burn' : The number of burned simulation in the start of the M-H algorithm. Default is 4000.
 - 'chains' : The number of M-H chains to run. Default is 1. If 'parallel' is set to true, this also will be the number of parallel workers.

- 'covrepair' : Give true to secure that the provided covariance matrix is positive semi-definite. If false (default), then an error will be thrown if the covariance matrix is not positive definite.
- 'draws' : The number of draws returned for each chain. As a scalar integer. Default is 10000.
- 'lb' : Lower bound on the variables. As a 1 x numVar double. Default is [], i.e. no lower bound on any variable.
- 'log' : Give true if the objective function is in log. Default is false.
- 'minIter' : A user defined threshold value that can be used by the function given by the 'qFunction' input.
 - If 'qFunction' is set to
 - > 'arw' : An 'adaptive' is set to 'recursive' this options sets how many past burning simulation that must be done before the calculation of the covariance matrix used by the 'phi' input does not use the burn in simulations.

Default is 20.
- 'parallel' : Set to true to run M-H in parallel using parfor.
- 'phi' : A function handle that simulates eps from the wanted distribution, i.e. the distribution to draw from when simulating the jump in the random variables at a iteration of the Metropolis - Hastings algorithm. Default is @(x)nb_mcmc.mvnRandChol(x).
- Caution : The x input is the same struct that is given by the output.
- 'qFuncPDF' : The PDF of the suggested conditional distribution that is assign to the 'qFunction' input. Only needed if 'qFunction' is not symmetric. If provided it must be a function handle that takes the output struct as its only input.

This function must provide 2 outputs. The first is the value of qFunction(betaDraw,betaLast), and the second is the value of qFunction(betaLast,betaDraw).
- 'qFunction' : A string or a function handle with how to draw the candidate distribution conditional on past draw. Either:
 - > 'rw' : Random-walk. $\beta(i+1) = \beta(i) + \text{eps}$, where $\text{eps} \sim \phi(0, \sigma)$.
 - > 'arw' : Adaptive random-walk. $\beta(i+1) = \beta(i) + \text{eps}$, where $\text{eps} \sim \phi(0, \sigma(i))$. In this case $\sigma(i)$ is an estimate of the covariance matrix of the sampled values of β .
 - > handle : A function handle taking 1 input. The input is on the same format as the output of this function.

Default is to use 'rw'.

This function must return 2 outputs; the candidate draw of the random variables with same size as the beta input, and it must return the input given to this function, as you may have alter it when constructing the candidate draw. For examples see; nb_mcmc.randomWalk or nb_mcmc.adaptiveRandomWalk.

- 'storeAt' : The number of simulations before the draws are stored to a file during the M-H algorithm. Default is 1000000.

Caution: In this case the names of the files that the draws are in can be found in the field files, while the beta output will be empty.
- 'storeFile' : Prefix of the names of the .mat files stored with the posterior draws. Include path if wanted. Caution: should not end with \. Defualt is 'mh_sampler'.
- 'thin' : The number of simulations before a draw is kept during the M-H algorithm. Default is 10.
- 'ub' : Upper bound on the variables. As a 1 x numVar double. Default is [], i.e. no lower bound on any variable.
- 'waitbar' : Give true to include waitbar.
- 'weight' : The weight put on the initial covariance matrix when 'qFunction' is set to 'arw' and 'adaptive' is set to 'recursive'. Default is 0.05 (the smalles possible value).

Output:

- output : A 1 x number of chains struct array with the following fields
 - > beta : A draws x numVar double. Empty if 'storeAt' is so small that the sampler start storing to files.
 - > files : The names of the files for where the draws are saved in the case tha the 'storeAt' limit is reached.

See also:

[nb_mcmc.randomWalk](#), [nb_mcmc.adaptiveRandomWalk](#)
[nb_mcmc.adaptiveRandomWalkUpdate](#), [nb_mcmc.geweke](#), [nb_mcmc.gelmanRubin](#)
[nb_mcmc.nutSampler](#), [nb_waitbar](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mcmc.mvnRand** ↑

betaDraw = nb_mcmc.mvnRand(output)

Description:

A function that may be assign to the 'phi' input to the nb_mcmc.mhSampler function.

Input:

- output : See documentation of the output from the nb_mcmc.mhSampler function.

Output:

- betaDraw : A nVar x 1 double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mcmc.mvnRandChol** ↑

betaDraw = nb_mcmc.mvnRandChol(output)

Description:

Default function assign to the 'phi' input to the nb_mcmc.mhSampler.

Input:

- output : See documentation of the output from the nb_mcmc.mhSampler function.

Output:

- betaDraw : A nVar x 1 double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mcmc.nutSampler** ↑

output = nb_mcmc.nutSampler(objective,beta,sigma,varargin)

Description:

Implementation of the No U-turn (NUT) sampler algorithm.

This uses the functions nb_mcmc.NUTS and nb_mcmc.dualAveraging made by Matthew D. Hoffman.

Input:

- objective : A function handle that represents a function that is proportional to the target distribution. Takes one input, and that are the simulated random variables, which has the same type and size as the beta input.

Caution : This function may also return the gradient of the objective. It must have size numVar x 1.
- beta : Initial values of the random variables to draw from. E.g. the mode found under optimization in the case of parameter estimates from a DSGE model. As a 1 x numVar double.
- sigma : Not used. Only to make it generic to other samplers...

Optional input:

- 'accTarget' : Sets the preferred acceptance ratio when 'adaptive' is set to 'target' or 'recTarget'. Default is 0.8.
- 'burn' : The number of burned simulation in the start of the M-H algorithm. Default is 4000.
- 'chains' : The number of M-H chains to run. Default is 1. If 'parallel' is set to true, this also will be the number of parallel workers.
- 'draws' : The number of draws returned for each chain. As a scalar integer. Default is 10000.
- 'lb' : Lower bound on the variables. As a 1 x numVar double. Default is [], i.e. no lower bound on any variable.
- 'log' : Give true if the objective function is in log. Default is false.
- 'parallel' : Set to true to run M-H in parallel using parfor.
- 'thin' : The number of simulations between storing a draw during the NUT algorithm. Default is 10.
- 'ub' : Upper bound on the variables. As a 1 x numVar double. Default is [], i.e. no upper bound on any variable.
- 'waitbar' : Give true to include waitbar.

Output:

- output : A 1 x number of chains struct array with the following fields
> beta : A draws x numVar double.

See also:

[nb_mcmc.NUTS](#), [nb_mcmc.dualAveraging](#), [nb_mcmc.mhSampler](#), [nb_waitbar](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_mcmc.optimset** ↑

```
opt = nb_mcmc.optimset(varargin)
```

Description:

Default setting of the `nb_mcmc.mhSampler` and `nb_mcmc.nutsampler`. See these function for help on the different fields (i.e. the optional inputs).

Optional input:

- `varargin` : Field name, field value pairs. These can be used to correct the default values to your prefered choice.

Output:

- `opt` : a struct with the default options of the sampler

See also:

[nb_mcmc.mhSampler](#), [nb_mcmc.nutsampler](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_mcmc.randomWalk** ↑

```
[betaDraw,output] = nb_mcmc.randomWalk(output)
```

Description:

Draw one candidate during Metropolis-Hastings algorithm using the random walk assumption.

Input:

- `output` : See the output of the `nb_mcmc.mhSampler` function.

Output:

- `betaDraw` : A `nVar x 1` double.

See also:

[nb_mcmc.mhSampler](#)

Written by Kenneth Sætherhagen Paulsen

■ nb_copula

Go to: [Properties](#) | [Methods](#)

A class for constructing a multivariate distribution given marginal distributions of the separate variables and a correlation matrix using a gaussian copula.

Caution: Some of the methods utilizes the MATLAB statistics package.

Superclasses:

handle

Constructor:

```
obj = nb_copula(distr,varargin)
```

Input:

- distr : A 1xN vector of nb_distribution objects. N > 1.
- varargin : Property name and property value pair arguments. See the set method for an example. Type properties(obj) in the command line to get a list of supported properties, and type help nb_distribution.<propertyName> to get help on a specific property.

Output:

- obj : An object of class nb_copula

Examples:

```
distr = nb_distribution.initialize('type',{'normal','gamma'},...  
                                'parameters',{{6,2},{2,2}});  
obj = nb_copula(distr);
```

Written by Kenneth Sætherhagen Paulsen

Properties:

- distributions • numberOfDistributions • sigma • transformedSigma • type

- **distributions** ↑

The marginal distributions of the separate variables as a 1xN vector of nb_distribution objects.

- **numberOfDistributions** ↑

The number of distributions in the copula. As a scalar double.

- **sigma** ↑

The linear correlation matrix. As a NxN double with ones along the diagonal. Must be reordered accordingly to the property distributions. Default is uncorrelated marginals.

- **transformedSigma** ↑

nb_copula/transformedSigma is a property.

- **type** ↑

Set the transformation method to use to transform the rank correlation coefficients to the linear correlation coefficients.

- 'none' : If the correlation in the data is calculated based on the linear correlation coefficients. Default.
- 'spearman' : Uses the formula $\rho = 2 \cdot \sin(\sigma \cdot \pi / 6)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X, 'type', 'spearman')` function.
- 'kendall' : Uses the formula $\rho = \sin(\sigma \cdot \pi / 2)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X, 'type', 'kendall')` function.

The rank correlation coefficients are approximately invariant to the choice of marginal distribution used to transform the draws from the copula to the draws of the different marginals.

Methods:

- `cdf`
- `domain`
- `get`
- `kurtosis`
- `mean`
- `median`
- `mode`
- `pdf`
- `plot`
- `random`
- `set`
- `skewness`
- `std`
- `variance`

► **cdf** ↑

`f = cdf(obj, x, type)`

Description:

Evaluate the cdf of the given distribution at the value(s) `x`.

The value is only an approximation!

Input:

- obj : An object of class nb_copula
- x : A double size TxN, where N is the dimension of the multivariate distribution.
- type : See output f.

Output:

- f : Dependent on the type input:
 - > 'full' : A Tx1 double with the multivariate cumulative probabilities at the wanted points of the multivariate domain. Default.
 - > 'marginals' : A TxN double with the marginal cumulative probabilities at the wanted points of the multivariate domain.
 - > 'conditional' : Calculate the conditional probability for the multivariate probabilities at the wanted points of the multivariate domain. The variables to condition on are identified by the the distribution which the conditionalValue property is set to a number.

Written by Kenneth Sæterhagen Paulsen

► domain ↑

d = domain(obj)

Description:

Returns the full domain of the marginal distributions combined by the copula.

Input:

- obj : An object of class nb_copula
- out :
 - 0 : Returns the lower and upper bound of the distribution, if truncated does limits are returned. Default.
 - 1 : Returns the lower and upper bound of the distribution, if truncated does limits are not returned.
 - 2 : Returns the lower and upper bound of the distribution, if truncated it will only return the lower truncation limit.
 - 3 : Returns the lower and upper bound of the distribution,

if truncated it will only return the upper truncation limit.

Output:

- domain : A Nx2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property value of a nb_copula object.

Type properties(obj) in the command line to get a list of supported properties, and type help nb_copula.<propertyName> to get help on a specific property.

Input:

- obj : An object of class nb_copula
- propertyName : A string with the property name

Output:

- propertyValue : The return value of the wanted property.

Examples:

```
propertyValue = get(obj,'type');
```

Written by Kenneth S. Paulsen

► **kurtosis** ↑

```
x = kurtosis(obj)
```

Description:

Evaluate the kurtosis of the marginal distributions combined by the copula.

The kurtosis will be adjusted for bias. See the function kurtosis made by MATLAB inc.

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth SÃ¶terhagen Paulsen

► mean ↑

x = mean(obj)

Description:

Evaluate the mean of the marginal distributions combined by the copula.

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth SÃ¶terhagen Paulsen

► median ↑

x = median(obj)

Description:

Evaluate the median of the marginal distributions combined by the copula.

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth SÃ¶terhagen Paulsen

► **mode** ↑

```
x = mode(obj)
```

Description:

Evaluate the mode of the marginal distributions combined by the copula.

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth Sæterhagen Paulsen

► **pdf** ↑

```
f = pdf(obj,x,type)
```

Description:

Evaluate the pdf of the given distribution at the value(s) x.

To condition on a value for a given marginal distribution, see the conditionalValue property of nb_distribution.

Input:

- obj : An object of class nb_copula

- x : A double size TxN, where N is the dimension of the multivariate distribution.

Caution : If 'type' is set to 'conditional' N is equal to the number of multivariate distribution not conditioned on.

- type : See output f.

Output:

- f : Dependent on the type input:

> 'full' : A Tx1 double with the multivariate

```

probabilities at the wanted points of the
multivariate domain. Default.

> 'marginals'    : A TxN double with the marginal probabilities at
                     the wanted points of the multivariate domain.

> 'conditional' : Calculate the conditional probability for the
                     multivariate probabilities at the wanted points
                     of the multivariate domain. The variables
                     to condition on are identified by the
                     the distribution which the conditionalValue
                     property is set to a number.

> 'log'          : Log of the likelihood.

```

Written by Kenneth Sæterhagen Paulsen

► **plot** ↑

```
plotter = plot(obj,type)
```

Description:

Graph either the PDF or CDF of one of the marginal distribution

Input:

- obj : A nb_distribution object.
- x : The values to evaluate the distribution. x must be a Nx1 double.
If not provided or empty the full domain of the distribution(s) will be used
- type : A string with either 'pdf' (default) or 'cdf'
- index : The index of the marginal distribution to plot. Default is 1.

Output:

- plotter : A nb_graph_data object. If nargout is 0 the graph will be plotted automatically, otherwise you need to call the graph method on the plotter object.

Examples:

```
obj = nb_distribution
plot(obj)
```

Written by Kenneth Sæterhagen Paulsen

► random ↑

```
D = random(obj,nrow,npage)
```

Description:

Draw random numbers from the multivariate distribution represented by the nb_copula object. I.e. uses a gaussian copula to draw correlated draws from each marginal distribution.

Caution: If numel(obj) > 1 the number of marginal distribution of each copula must be the same.

Caution: If some of the marginal distributions objects has set the conditonalValue property to some number, the random numbers are generated conditional on this value(s).

Input:

- obj : An object of class nb_copula
- nrow : The number of rows of the draws output
- npage : The number of pages of the draws output

Output:

- D : A nrow x N x npage x nobj1 x nobj2 double with the draws from the multivariate distribution, where N is the number of marginal distributions.

Examples:

```
distr = nb_distribution.initialize('type',{'normal','gamma'},...  
                                    'parameters',{{6,2},{2,2}});  
obj   = nb_copula(distr,'type','kendall','sigma',[1,0.5;0.5,1]);  
D     = random(obj,100);
```

Written by Kenneth Sæterhagen Paulsen

► set ↑

```
set(obj,varargin)
```

Description:

Sets the properties of the nb_copula object.

Type properties(obj) in the command line to get a list of supported properties, and type help nb_copula.<propertyName> to get help on a specific property.

Input:

- obj : An object of class nb_copula
- varargin : 'propertyName',PropertyValue,...

Output:

- obj : The input object with the updated properties.

Examples:

```
set(obj,'propertyName',PropertyValue);
obj.set('propertyName',PropertyValue,...)
obj = obj.set('type','kendall');
```

Written by Kenneth S. Paulsen

► **skewness** ↑

```
x = skewness(obj)
```

Description:

Evaluate the skewness of the marginal distributions combined by the copula.

The skewness will be adjusted for bias. See the function skewness made by MATLAB inc.

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth Sæterhagen Paulsen

► **std** ↑

```
x = std(obj)
```

Description:

Evaluate the standard deviation of the marginal distributions combined by the copula.

The variance is normalized with T-1

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth Sæterhagen Paulsen

► **variance ↑**

x = variance(obj)

Description:

Evaluate the variance of the marginal distributions combined by the copula.

The variance is normalized with T-1

Input:

- obj : An object of class nb_copula

Output:

- x : A double with size 1xN, where N is the number of marginal distributions combined with the copula.

Written by Kenneth Sæterhagen Paulsen

■ **nb_distribution**

Go to: [Properties](#) | [Methods](#)

A class for evaluate the pdf, cdf, icdf etc of a known distribution as well as for a estimated distribution (using kernel density estimates)

Caution: Some of the methods utilizes the MATLAB statistics package.

Superclasses:

handle

Constructor:

```
obj = nb_distribution(varargin)
```

Input:

- varargin : Property name and property value pair arguments. See the set method for a example. Type properties(obj) in the command line to get a list of supported properties, and type help nb_distribution.<propertyName> to get help on a specific property.

Caution : When the property type is set, the parameters property will be set to its default value given the type of distribution. See the paramters for more on this.

Output:

- obj : An object of class nb_distribution

Examples:

```
obj = nb_distribution('type','gamma');
```

Written by Kenneth Sætherhagen Paulsen

Properties:

- conditionalValue • lowerBound • meanShift • name • parameters
- type • upperBound • userData

- **conditionalValue** ↑

The value to condition on when included in a nb_copula object and the 'conditional' type is used by the pdf, cdf and random methods, otherwise it has no effect.

- **lowerBound** ↑

Lower bound of truncated distribution. Use the set method to truncate the distribution.

- **meanShift** ↑

Shift mean. As a double. If the distribution is truncated this will only shift the untruncated distributions mean with this magnitude. Default is empty, i.e. no mean shift.

- **name** ↑

The name of the distribution. It is possible to set this property by the set method. If not set it will be given a default name that will depend on the distribution and its parametrization.

- **parameters** ↑

The parameters of the wanted distribution. This property will depend on the distribution type;

Caution : When the property type is set, the parameters property will be set to its default value given the type of distribution.

- 'ast' : The parameters property must be set to a 1x5 cell. {location,scale,skewness,lefthead,righttail}, where location is any double, scale > 0, skewness is between 0 and 1, lefthead > 0 and righttail > 0. Default is {0,1,0,1000,1000} ~ N(0,1).
- 'beta' : The parameters property must be set to a 1x2 cell. {param1,param2}, where both must be set to a 1x1 double greater than 0. Default is {2,2}. The mean will be given by param1/(param1 + param2).
- 'cauchy' : The parameters property must be set to a 1x2 cell. {location,scale}, where location must be set to a 1x1 double, and scale must be set to a 1x1 double greater than 0. Default is {0,2}.
- 'chis' : The parameters property must be set to a 1x1 cell. {param1}, where param1 must be set to a 1x1 double greater than 0. Default is {2}.
- 'constant' : The parameters property must be set to a 1x1 cell. {constant}, where constant must be set to a 1x1 double. Default is {0}.
- 'empirical' : The parameters property must be set to a 1 x 2 cell. First element must be the domain (1 x N double) and the second element must be the probability density (1 x N double).
- 'exp' : The parameters property must be set to a 1x1 cell. {param1}, where param1 must be set to a 1x1 double greater than 0. Default is {2}.
- 'f' : The parameters property must be set to a 1x1 cell. {param1,parma2}, where both must be set to a 1x1 double greater than 0. Default is {10,10}.
- 'fgamma' : The parameters property must be set to a 1x2 cell. {shape,scale}, where shape and scale must both be a 1x1 double greater than 0. Default is {2,2}. The mean will be given by shape*scale.
- 'finvgamma' : The parameters property must be set to a 1x2 cell. {shape,scale}, where shape and scale must both be a 1x1 double greater than 0. Default is {4,2}. The mean will be given by scale/(shape - 1) for shape > 1.
- 'gamma' : The parameters property must be set to a 1x2 cell. {shape,scale}, where shape and scale must both be a 1x1 double greater than 0. Default is {2,2}. The mean will be given by shape*scale.

- 'hist' : The paramters property must be set to a 1x1 cell. {data}, where data must be a Nx1 double. Default is {randn(100,1)}.
 - 'invgamma' : The paramters property must be set to a 1x2 cell. {shape,scale}, where shape and scale must both be a 1x1 double greater than 0. Default is {4,2}. The mean will be given by scale/(shape - 1) for shape > 1.
 - 'kernel' : The parameters property must be set to a 1 x 2 cell. First element must be the domain (1 x N double) and the second element must be the probability density (1 x N double).
 - 'laplace' : The paramters property must be set to a 1x2 cell. {location,scale}, where location must be a 1x1 double and scale must be 1x1 double greater than 0. Default is {0,1}.
 - 'logistic' : The paramters property must be set to a 1x2 cell. {location,scale}, where location must be set to a 1x1 double, and scale must be set to a 1x1 double greater than 0. Default is {0,2}.
 - 'lognormal' : The paramters property must be set to a 1x2 cell. {param1,param2}, where both must be set to a 1x1 double greater than 0. Default is {1,1}.
 - 'normal' : The paramters property must be set to a 1x2 cell. {mean,std}, where mean must be a 1x1 double and std must be 1x1 double greater than 0. Default is {0,1}.
 - 'skewedt' : The paramters property must be set to a 1x4 cell. {mean,std,skewness,kurtosis}. Default is {0,1,2.910,0,2.5}, i.e. same as student-t with 5 degrees of freedom.
 - 'skt' : The paramters property must be set to a 1x4 cell. {location,scale,shape,dof}. Default is {0,1,0,inf}, i.e. same as the normal distribution. dof is the degree of freedom, and must be set to a 1x1 double greater than 0. shape must be in (-1,1). scale > 0. No restrictions on location.
 - 't' : The paramters property must be set to a 1x1 cell. {dof}, where dof must be set to a 1x1 double greater than 0. Default is {5}.
- You can also use the Non-standardized student-t distribution by setting this property to a 1x3 cell, where the two additional inputs are location and scale. location must be set to a 1x1 double and scale must be set to a 1x1 double greater than 0.
- 'tri' : The paramters property must be set to a 1x3 cell.

- {low,high,mode}, where low, high and mode must be set to a 1x1 double. Default is {0,1,0.5}.
- 'uniform' : The paramters property must be set to a 1x2 cell. {lower,upper}, where bust must be a 1x1 double. Default is {0,1}.
 - 'wald' : The paramters property must be set to a 1x2 cell. {mean,shape}, where mean must be a 1x1 double greater than 0 and shape must be 1x1 double greater than 0. Default is {1,1}.
- type ↑**
- The type of distributions. The following are supported;
- 'ast' : Asymetric student-t distribution. See Zhu and Galbraith (2009)
 - 'beta' : Beta distribution
 - 'cauchy' : Cauchy distribution
 - 'chis' : Chi-squared distribution
 - 'constant' : A constant. I.e. the distribution of a non-random variable.
 - 'empirical' : Empirical (Kaplan-Meier) cumulative distribution function.
 - 'f' : F-distribution
 - 'fgamma' : Flipped gamma distribution
 - 'finvgamma' : Flipped inverse gamma distribution
 - 'gamma' : Gamma distribution
 - 'hist' : Create a histogram. This to make it easy to compare the estimated distribution against a histogram. For the moments operators the empirical counterparts are returned!
 - 'invgamma' : Inverse gamma distribution
 - 'kernel' : Kernel density estimation. Please see the method estimate on how to estimate a kernel distribution.
 - 'laplace' : Laplace distribution.
 - 'lognormal' : Lognormal distribution.
 - 'logistic' : Logistic distribution.
 - 'normal' : Normal distribution. Default
 - 'skewedt' : Skewed t-distribution. [https://en.wikipedia.org/...](https://en.wikipedia.org/)

wiki/Skewed_generalized_t_distribution

- 'skt' : Azzalini skewed t-distribution. Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution.
J.Roy.Statist.Soc. B 65, 367–389. May not be available in public version.
- 't' : Student-t distribution
- 'tri' : Triangular distribution
- 'uniform' : Uniform distribution
- 'wald' : Wald distribution

- **upperBound** ↑

Upper bound of truncated distribution. Use the set method to truncate the distribution.

- **userData** ↑

User data stored to the object. Can be of any type

Methods:

- asData
- assignParameters
- ast_cdf
- ast_domain
- ast_icdf
- ast_kurtosis
- ast_mean
- ast_median
- ast_mode
- ast_pdf
- ast_rand
- ast_skewness
- ast_std
- ast_variance
- beta_cdf
- beta_domain
- beta_icdf
- beta_kurtosis
- beta_mean
- beta_median
- beta_mode
- beta_pdf
- beta_rand
- beta_skewness
- beta_std
- beta_variance
- cauchy_cdf
- cauchy_domain
- cauchy_icdf
- cauchy_kurtosis
- cauchy_mean
- cauchy_median
- cauchy_mode
- cauchy_pdf
- cauchy_rand
- cauchy_skewness
- cauchy_std
- cauchy_variance
- cdf
- chis_cdf
- chis_domain
- chis_icdf
- chis_kurtosis
- chis_mean
- chis_median
- chis_mode
- chis_pdf
- chis_rand
- chis_skewness
- chis_std
- chis_variance
- confidenceInterval
- constant_cdf
- constant_domain

- constant_icdf
- constant_mean
- constant_mode
- constant_rand
- constant_std
- convert
- domain
- double2NormalDist
- empirical_domain
- empirical_mean
- empirical_mode
- empirical_rand
- empirical_std
- estimate
- exp_cdf
- exp_icdf
- exp_mean
- exp_mode
- exp_rand
- exp_std
- f_cdf
- f_icdf
- f_mean
- f_mode
- f_rand
- f_std
- fgamma_cdf
- constant_kurtosis
- constant_median
- constant_pdf
- constant_skewness
- constant_variance
- defaultParameters
- double2Dist
- empirical_cdf
- empirical_icdf
- empirical_median
- empirical_pdf
- empirical_skewness
- empirical_variance
- estimateDomain
- exp_domain
- exp_kurtosis
- exp_median
- exp_pdf
- exp_skewness
- exp_variance
- f_domain
- f_kurtosis
- f_median
- f_pdf
- f_skewness
- f_variance
- fgamma_domain

- `fgamma_icdf`
- `fgamma_mean`
- `fgamma_mode`
- `fgamma_rand`
- `fgamma_std`
- `finvgamma_cdf`
- `finvgamma_icdf`
- `finvgamma_mean`
- `finvgamma_mode`
- `finvgamma_rand`
- `finvgamma_std`
- `gamma_cdf`
- `gamma_icdf`
- `gamma_mean`
- `gamma_mode`
- `gamma_rand`
- `gamma_std`
- `get`
- `getNumberOfParameters`
- `getStartOfDomain`
- `hist_cdf`
- `hist_icdf`
- `hist_mean`
- `hist_mode`
- `hist_rand`
- `hist_std`
- `horzcat`
- `fgamma_kurtosis`
- `fgamma_median`
- `fgamma_pdf`
- `fgamma_skewness`
- `fgamma_variance`
- `finvgamma_domain`
- `finvgamma_kurtosis`
- `finvgamma_median`
- `finvgamma_pdf`
- `finvgamma_skewness`
- `finvgamma_variance`
- `gamma_domain`
- `gamma_kurtosis`
- `gamma_median`
- `gamma_pdf`
- `gamma_skewness`
- `gamma_variance`
- `getEndOfDomain`
- `getParameters`
- `getTheoreticalHistCounts`
- `hist_domain`
- `hist_kurtosis`
- `hist_median`
- `hist_pdf`
- `hist_skewness`
- `hist_variance`
- `icdf`

- increment
- invgamma_cdf
- invgamma_icdf
- invgamma_mean
- invgamma_mode
- invgamma_rand
- invgamma_std
- invwish_rand
- kernel_cdf
- kernel_icdf
- kernel_mean
- kernel_mode
- kernel_rand
- kernel_std
- kurtosis
- laplace_domain
- laplace_kurtosis
- laplace_median
- laplace_pdf
- laplace_skewness
- laplace_variance
- logistic_domain
- logistic_kurtosis
- logistic_median
- logistic_pdf
- logistic_skewness
- logistic_variance
- initialize
- invgamma_domain
- invgamma_kurtosis
- invgamma_median
- invgamma_pdf
- invgamma_skewness
- invgamma_variance
- isnan
- kernel_domain
- kernel_kurtosis
- kernel_median
- kernel_pdf
- kernel_skewness
- kernel_variance
- laplace_cdf
- laplace_icdf
- laplace_mean
- laplace_mode
- laplace_rand
- laplace_std
- logistic_cdf
- logistic_icdf
- logistic_mean
- logistic_mode
- logistic_rand
- logistic_std
- lognormal_cdf

- lognormal_domain
- lognormal_kurtosis
- lognormal_median
- lognormal_pdf
- lognormal_skewness
- lognormal_variance
- meanshift_cdf
- meanshift_icdf
- meanshift_mean
- meanshift_mode
- meanshift_rand
- meanshift_std
- median
- mme
- normal_cdf
- normal_icdf
- normal_mean
- normal_mode
- normal_rand
- normal_std
- parameterization
- pdf
- perc2DistCDF
- percentile
- plot3d
- random
- sim2KernelDist
- lognormal_icdf
- lognormal_mean
- lognormal_mode
- lognormal_rand
- lognormal_std
- mean
- meanshift_domain
- meanshift_kurtosis
- meanshift_median
- meanshift_pdf
- meanshift_skewness
- meanshift_variance
- mle
- mode
- normal_domain
- normal_kurtosis
- normal_median
- normal_pdf
- normal_skewness
- normal_variance
- parametrization
- perc2Dist
- perc2ParamDist
- plot
- qestimation
- set
- skewedt_cdf

- skewedt_domain
- skewedt_kurtosis
- skewedt_median
- skewedt_pdf
- skewedt_skewness
- skewedt_variance
- skt_cdf
- skt_icdf
- skt_mean
- skt_mode
- skt_rand
- skt_std
- smoothDensity
- t_cdf
- t_icdf
- t_mean
- t_mode
- t_rand
- t_std
- tri_cdf
- tri_icdf
- tri_mean
- tri_mode
- tri_rand
- tri_std
- truncated_cdf
- truncated_icdf
- skewedt_icdf
- skewedt_mean
- skewedt_mode
- skewedt_rand
- skewedt_std
- skewness
- skt_domain
- skt_kurtosis
- skt_median
- skt_pdf
- skt_skewness
- skt_variance
- std
- t_domain
- t_kurtosis
- t_median
- t_pdf
- t_skewness
- t_variance
- tri_domain
- tri_kurtosis
- tri_median
- tri_pdf
- tri_skewness
- tri_variance
- truncated_domain
- truncated_kurtosis

- truncated_mean
 - truncated_mode
 - truncated_rand
 - truncated_std
 - truncgamma_mean
 - truncnormal_mean
 - uniform_cdf
 - uniform_icdf
 - uniform_mean
 - uniform_mode
 - uniform_rand
 - uniform_std
 - var
 - vertcat
 - wald_domain
 - wald_kurtosis
 - wald_median
 - wald_pdf
 - wald_skewness
 - wald_variance
 - truncated_median
 - truncated_pdf
 - truncated_skewness
 - truncated_variance
 - truncgamma_variance
 - truncnormal_variance
 - uniform_domain
 - uniform_kurtosis
 - uniform_median
 - uniform_pdf
 - uniform_skewness
 - uniform_variance
 - variance
 - wald_cdf
 - wald_icdf
 - wald_mean
 - wald_mode
 - wald_rand
 - wald_std
 - wish_rand
-

► **asData** ↑

```
data = asData(obj,x,type)
```

Description:

Create an nb_data object from either the PDF or CDF of the distribution

Input:

- obj : A nb_distribution object.
- x : The values to evaluate the distribution. x must be a Nx1 double.
If not provided or empty the full domain of the distribution(s) will be used
- type : A string with either 'pdf' (default) or 'cdf'

Output:

- data : A nb_data object.

Examples:

```
obj = nb_distribution  
asData(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **assignParameters ↑**

```
assignParameters(obj,param)
```

Description:

Assign parameters to a scalar distribution object.

Input:

- obj : A nb_distribution object.
- param : The parameters to assign the object, as 1 x nParam double.

Written by Kenneth Sæterhagen Paulsen

► **ast_cdf ↑**

```
f = nb_distribution.ast_cdf(x,a,b,c,d,e)
```

Description:

CDF of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- x : The point to evaluate the cdf, as a double.
- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter (1>c>0).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.ast_pdf](#), [nb_distribution.ast_rand](#)
[nb_distribution.ast_icdf](#), [nb_mci](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_domain** ↑

f = nb_distribution.ast_domain

Description:

Get the domain of the asymmetric t-distribution of Zhu and Galbraith (2009).

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **ast_icdf** ↑

x = nb_distribution.ast_icdf(p,a,b,c,d,e)

Description:

Returns the inverse of the cdf at p of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- p : A vector of probabilities
- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter (1>c>0).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : A double with the quantile at each element of p of the gamma(m,k) distribution

See also:

[nb_distribution.ast_cdf](#), [nb_distribution.ast_rand](#)
[nb_distribution.ast_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_kurtosis ↑**

x = nb_distribution.ast_kurtosis(a,b,c,d,e)

Description:

Kurtosis of the asymmetric t-distribution of Zhu and Galbraith (2009).

Caution : Simulation based (seed is set) !

Input:

- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter (1>c>0).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The kurtosis of the distribution

See also:

[nb_distribution.ast_median](#), [nb_distribution.ast_mean](#)
[nb_distribution.ast_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_mean ↑**

x = nb_distribution.ast_mean(a,b,c,d,e)

Description:

Mean of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter ($1 > c > 0$).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The mean of the distribution

See also:

[nb_distribution.ast_mode](#), [nb_distribution.ast_median](#)
[nb_distribution.ast_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_median ↑**

x = nb_distribution.ast_median(a,b,c,d,e)

Description:

Median of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The skewness parameter.
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The median of the distribution

See also:

[nb_distribution.ast_mode](#), [nb_distribution.ast_mean](#)
[nb_distribution.ast_variance](#)

Written by Kenneth Å!terhagen Paulsen

► **ast_mode** ↑

```
x = nb_distribution.ast_mode(a,b,c,d,e)
```

Description:

Mode of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter ($1 > c > 0$).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The mode of the distribution

See also:

[nb_distribution.ast_median](#), [nb_distribution.ast_mean](#)
[nb_distribution.ast_variance](#)

Written by Kenneth Å!terhagen Paulsen

► **ast_pdf** ↑

```
f = nb_distribution.ast_pdf(x,a,b,c,d,e)
```

Description:

PDF of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- `x` : The point to evaluate the pdf, as a double
- `a` : The location parameter.
- `b` : The scale parameter (>0).
- `c` : The skewness parameter ($1>c>0$).
- `d` : The left parameter (>0).
- `e` : The right parameter (>0).

Output:

- `f` : The PDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.ast_cdf](#), [nb_distribution.ast_rand](#)
[nb_distribution.ast_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_rand** ↑

```
draws = nb_distribution.ast_rand(nrow, ncol, a, b, c, d, e)
```

Description:

Draw random numbers from the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- `nrow` : Number of rows of the matrix drawn from the distribution
- `ncol` : Number of columns of the matrix drawn from the distribution
- `a` : The location parameter.
- `b` : The scale parameter (>0).
- `c` : The skewness parameter ($1>c>0$).
- `d` : The left parameter (>0).
- `e` : The right parameter (>0).

Output:

- `draws` : A `nrow` x `ncol` matrix of random numbers from the distribution

See also:

[nb_distribution.ast_pdf](#), [nb_distribution.ast_cdf](#)

Modified by Kenneth S. Paulsen

► **ast_skewness ↑**

`x = nb_distribution.ast_skewness(a,b,c,d,e)`

Description:

Skewness of the asymmetric t-distribution of Zhu and Galbraith (2009).

Caution : Simulation based (seed is set) !

Input:

- `a` : The location parameter.
- `b` : The scale parameter (>0).
- `c` : The skewness parameter ($1>c>0$).
- `d` : The left parameter (>0).
- `e` : The right parameter (>0).

Output:

- `x` : The skewness of the distribution

See also:

[nb_distribution.ast_median](#), [nb_distribution.ast_mean](#)
[nb_distribution.ast_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **ast_std ↑**

`x = nb_distribution.ast_std(a,b,c,d,e)`

Description:

Standard deviation of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter ($1>c>0$).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The standard deviation of the distribution

See also:

[nb_distribution.ast_mode](#), [nb_distribution.ast_median](#)
[nb_distribution.ast_mean](#), [nb_distribution.ast_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [ast_variance ↑](#)

`x = nb_distribution.ast_variance(a,b,c,d,e)`

Description:

Variance of the asymmetric t-distribution of Zhu and Galbraith (2009).

Input:

- a : The location parameter.
- b : The scale parameter (>0).
- c : The skewness parameter ($1>c>0$).
- d : The left parameter (>0).
- e : The right parameter (>0).

Output:

- x : The variance of the distribution

See also:

[nb_distribution.ast_mode](#), [nb_distribution.ast_median](#)
[nb_distribution.ast_mean](#)

► **beta_cdf** ↑

```
f = nb_distribution.beta_cdf(x,m,k)
```

Description:

CDF of the beta distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.beta_pdf](#), [nb_distribution.beta_rand](#)
[nb_distribution.beta_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_domain** ↑

```
f = nb_distribution.beta_domain
```

Description:

Get the domain of the beta distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **beta_icdf** ↑

```
x = nb_distribution.beta_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the beta(m,k) distribution

Input:

- p : A vector of probabilities
- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : A double with the quantile at each element of p of the beta(m,k) distribution

See also:

[nb_distribution.beta_cdf](#), [nb_distribution.beta_rand](#)
[nb_distribution.beta_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_kurtosis ↑**

```
x = nb_distribution.beta_kurtosis(m,k)
```

Description:

Kurtosis of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The kurtosis of the beta distribution

See also:

[nb_distribution.beta_median](#), [nb_distribution.beta_mean](#)
[nb_distribution.beta_variance](#)

► **beta_mean** ↑

```
x = nb_distribution.beta_mean(m,k)
```

Description:

Mean of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The mean of the beta distribution

See also:

[nb_distribution.beta_mode](#), [nb_distribution.beta_median](#)
[nb_distribution.beta_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_median** ↑

```
x = nb_distribution.beta_median(m,k)
```

Description:

Median of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The median of the beta distribution

See also:

[nb_distribution.beta_mode](#), [nb_distribution.beta_mean](#)
[nb_distribution.beta_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_mode** ↑

```
x = nb_distribution.beta_mode(m, k)
```

Description:

Mode of the beta distribution. When the $m, k \leq 1$ the distribution get bimodal, so mode can be either 0 or 1, so nan is returned

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The mode of the beta distribution

See also:

[nb_distribution.beta_median](#), [nb_distribution.beta_mean](#)
[nb_distribution.beta_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_pdf** ↑

```
f = nb_distribution.beta_pdf(x, m, k)
```

Description:

PDF of the beta distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.beta_cdf](#), [nb_distribution.beta_rand](#)
[nb_distribution.beta_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_rand** ↑

```
draws = nb_distribution.beta_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the beta distribution

Input:

- nrow : Number of rows of the matrix drawn from the beta distribution
- ncol : Number of columns of the matrix drawn from the beta distribution
- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- draws : A nrow x ncol matrix of random numbers from the beta distribution

See also:

[nb_distribution.beta_pdf](#), [nb_distribution.beta_cdf](#)

Written by Kenneth S. Paulsen

► **beta_skewness** ↑

```
x = nb_distribution.beta_skewness(m, k)
```

Description:

Skewness of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The skewness of the beta distribution

See also:

[nb_distribution.beta_median](#), [nb_distribution.beta_mean](#)
[nb_distribution.beta_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_std ↑**

x = nb_distribution.beta_std(m, k)

Description:

Standard deviation of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The standard deviation of the beta distribution

See also:

[nb_distribution.beta_mode](#), [nb_distribution.beta_median](#)
[nb_distribution.beta_mean](#), [nb_distribution.beta_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **beta_variance ↑**

```
x = nb_distribution.beta_variance(m,k)
```

Description:

Variance of the beta distribution

Input:

- m : First shape parameter of the beta distribution. The mean will be given by $m/(m + k)$
- k : Second shape parameter of the beta distribution

Output:

- x : The variance of the beta distribution

See also:

[nb_distribution.beta_mode](#), [nb_distribution.beta_median](#)
[nb_distribution.beta_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_cdf** ↑

```
f = nb_distribution.cauchy_cdf(x,m)
```

Description:

CDF of the cauchy distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.cauchy_pdf](#), [nb_distribution.cauchy_rand](#)
[nb_distribution.cauchy_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_domain** ↑

```
f = nb_distribution.cauchy_domain
```

Description:

Get the domain of the cauchy distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **cauchy_icdf** ↑

```
x = nb_distribution.cauchy_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the cauchy distribution

Input:

- p : A vector of probabilities
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : A double with the quantile at each element of p of the cauchy(m,k) distribution

See also:

[nb_distribution.cauchy_cdf](#), [nb_distribution.cauchy_rand](#)
[nb_distribution.cauchy_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_kurtosis** ↑

```
x = nb_distribution.cauchy_kurtosis(m,k)
```

Description:

Kurtosis of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The kurtosis of the cauchy distribution

See also:

[nb_distribution.cauchy_median](#), [nb_distribution.cauchy_mean](#)
[nb_distribution.cauchy_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_mean** ↑

x = nb_distribution.cauchy_mean(m, k)

Description:

Mean of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mean of the cauchy distribution

See also:

[nb_distribution.cauchy_mode](#), [nb_distribution.cauchy_median](#)
[nb_distribution.cauchy_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_median** ↑

```
x = nb_distribution.cauchy_median(m, k)
```

Description:

Median of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The median of the cauchy distribution

See also:

[nb_distribution.cauchy_mode](#), [nb_distribution.cauchy_mean](#)
[nb_distribution.cauchy_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_mode** ↑

```
x = nb_distribution.cauchy_mode(m, k)
```

Description:

Mode of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mode of the cauchy distribution

See also:

[nb_distribution.cauchy_median](#), [nb_distribution.cauchy_mean](#)
[nb_distribution.cauchy_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_pdf** ↑

```
f = nb_distribution.cauchy_pdf(x,m,k)
```

Description:

PDF of the cauchy distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.cauchy_cdf](#), [nb_distribution.cauchy_rand](#)
[nb_distribution.cauchy_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_rand** ↑

```
draws = nb_distribution.cauchy_rand(nrow,ncol,m,k)
```

Description:

Draw random numbers from the cauchy distribution

Input:

- nrow : Number of rows of the matrix drawn from the cauchy distribution
- ncol : Number of columns of the matrix drawn from the cauchy distribution
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

```
- draws : A nrow x ncol matrix of random numbers from the cauchy distribution
```

See also:

[nb_distribution.cauchy_pdf](#), [nb_distribution.cauchy_cdf](#)

Modified by Kenneth S. Paulsen

► **cauchy_skewness ↑**

```
x = nb_distribution.cauchy_skewness(m, k)
```

Description:

Skewness of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The skewness of the cauchy distribution

See also:

[nb_distribution.cauchy_median](#), [nb_distribution.cauchy_mean](#)
[nb_distribution.cauchy_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **cauchy_std ↑**

```
x = nb_distribution.cauchy_std(m, k)
```

Description:

Standard deviation of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The standard deviation of the cauchy distribution

See also:

`nb_distribution.cauchy_mode`, `nb_distribution.cauchy_median`
`nb_distribution.cauchy_mean`, `nb_distribution.cauchy_variance`

Written by Kenneth Sæterhagen Paulsen

► **cauchy_variance ↑**

`x = nb_distribution.cauchy_variance(m, k)`

Description:

Variance of the cauchy distribution

Input:

- m : The location parameter. Mode = m. Must be a number.

- k : Scale parameter. Must be a positive number.

Output:

- x : The variance of the cauchy distribution

See also:

`nb_distribution.cauchy_mode`, `nb_distribution.cauchy_median`
`nb_distribution.cauchy_mean`

Written by Kenneth Sæterhagen Paulsen

► **cdf ↑**

`f = cdf(obj, x)`

Description:

Evaluate the cdf of the given distribution at the value(s) x.

Input:

- `obj` : An object of class `nb_distribution`
- `x` : A double with any size if `numel(obj) == 1`, otherwise it must be a `Tx1` double.

Output:

- `f` : `numel(obj) == 1` : A double with the same size as `x`
otherwise : A double with size `T x nobj`

Written by Kenneth Sæterhagen Paulsen

► **chis_cdf** ↑

```
f = nb_distribution.chis_cdf(x,m)
```

Description:

CDF of the chi squared distribution

Input:

- `x` : The point to evaluate the cdf, as a double
- `m` : A parameter such that the mean is `m`

Output:

- `f` : The CDF at the evaluated points, same size as `x`.

See also:

`nb_distribution.chis_pdf`, `nb_distribution.chis_rand`
`nb_distribution.chis_icdf`

Written by Kenneth Sæterhagen Paulsen

► **chis_domain** ↑

```
f = nb_distribution.chis_domain
```

Description:

Get the domain of the chis distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **chis_icdf** ↑

```
x = nb_distribution.chis_icdf(p,m)
```

Description:

Returns the inverse of the cdf at p of the chi squared distribution

Input:

- p : A vector of probabilities
- m : A parameter such that the mean is m

Output:

- x : A double with the quantile at each element of p of the chis(m,k) distribution

See also:

[nb_distribution.chis_cdf](#), [nb_distribution.chis_rand](#)
[nb_distribution.chis_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **chis_kurtosis** ↑

```
x = nb_distribution.chis_kurtosis(m)
```

Description:

Kurtosis of the chi squared distribution

Input:

- m : A parameter such that the mean is m

Output:

- x : The kurtosis of the chi squared distribution

See also:

`nb_distribution.chis_median`, `nb_distribution.chis_mean`
`nb_distribution.chis_variance`

Written by Kenneth Sæterhagen Paulsen

► **chis_mean** ↑

`x = nb_distribution.chis_mean(m)`

Description:

Mean of the chi squared distribution

Input:

- `m` : A parameter such that the mean is `m`

Output:

- `x` : The mean of the chi squared distribution

See also:

`nb_distribution.chis_mode`, `nb_distribution.chis_median`
`nb_distribution.chis_variance`

Written by Kenneth Sæterhagen Paulsen

► **chis_median** ↑

`x = nb_distribution.chis_median(m)`

Description:

Median of the chi squared distribution

Input:

- `m` : A parameter such that the mean is `m`

Output:

- `x` : The median of the chi squared distribution

See also:

`nb_distribution.chis_mode`, `nb_distribution.chis_mean`

[nb_distribution.chis_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **chis_mode** ↑

`x = nb_distribution.chis_mode(m)`

Description:

Mode of the chi squared distribution

Input:

- `m` : A parameter such that the mean is `m`

Output:

- `x` : The mode of the chi squared distribution

See also:

[nb_distribution.chis_median](#), [nb_distribution.chis_mean](#)
[nb_distribution.chis_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **chis_pdf** ↑

`f = nb_distribution.chis_pdf(x,m)`

Description:

PDF of the chi squared distribution

Input:

- `x` : The point to evaluate the pdf, as a double

- `m` : A parameter such that the mean is `m`

Output:

- `f` : The PDF at the evaluated points, same size as `x`.

See also:

`nb_distribution.chis_cdf`, `nb_distribution.chis_rand`
`nb_distribution.chis_icdf`

Written by Kenneth SÃ¶terhagen Paulsen

► **chis_rand** ↑

`draws = nb_distribution.chis_rand(nrow, ncol, m)`

Description:

Draw random numbers from the chis distribution

Input:

- `nrow` : Number of rows of the matrix drawn from the chis distribution
- `ncol` : Number of columns of the matrix drawn from the chis distribution
- `m` : A parameter such that the mean is `m`

Output:

- `draws` : A `nrow x ncol` matrix of random numbers from the chis distribution

See also:

`nb_distribution.chis_pdf`, `nb_distribution.chis_cdf`

Modified by Kenneth S. Paulsen

► **chis_skewness** ↑

`x = nb_distribution.chis_skewness(m)`

Description:

Skewness of the chi squared distribution

Input:

- `m` : A parameter such that the mean is `m`

Output:

- x : The skewness of the chi squared distribution

See also:

[nb_distribution.chis_median](#), [nb_distribution.chis_mean](#)
[nb_distribution.chis_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **chis_std** ↑

x = nb_distribution.chis_std(m)

Description:

Standard deviation of the chi squares distribution

Input:

- m : A parameter such that the mean is m

Output:

- x : The standard deviation of the chi squared distribution

See also:

[nb_distribution.chis_mode](#), [nb_distribution.chis_median](#)
[nb_distribution.chis_mean](#), [nb_distribution.chis_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **chis_variance** ↑

x = nb_distribution.chis_variance(m)

Description:

Variance of the chi squared distribution

Input:

- m : A parameter such that the mean is m

Output:

- x : The variance of the chi squared distribution

See also:

[nb_distribution.chis_mode](#), [nb_distribution.chis_median](#)
[nb_distribution.chis_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **confidenceInterval** ↑

```
ci = confidenceInterval(distribution, alpha)
```

Description:

Calculate confidence/probability interval

Input:

- distribution : A nb_distribution object
- alpha : Significance value. Default is 0.05;

Output:

- ci : A nobj x 2 double storing the confidence/probability intervals.

See also:

[nb_distribution.icdf](#)

Written by Henrik Halvorsen Hortemo

► **constant_cdf** ↑

```
f = nb_distribution.constant_cdf(x,m)
```

Description:

CDF of a constant.

Input:

- x : The point to evaluate the cdf, as a double
- m : The constant

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.constant_pdf](#), [nb_distribution.constant_rand](#)
[nb_distribution.constant_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **constant_domain** ↑

```
f = nb_distribution.constant_domain
```

Description:

Get the domain of a constant.

Input:

- m : The constant

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **constant_icdf** ↑

```
x = nb_distribution.constant_icdf(p,m)
```

Description:

Returns the inverse of the cdf at p of a constant

Input:

- p : A vector of probabilities
- m : The constant

Output:

- x : A double with the quantile at each element of p of the constant(m,k) distribution

See also:

[nb_distribution.constant_cdf](#), [nb_distribution.constant_rand](#)
[nb_distribution.constant_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **constant_kurtosis** ↑

`x = nb_distribution.constant_kurtosis(m)`

Description:

Kurtosis of a constant

Input:

- `m` : The constant

Output:

- `x` : 0

See also:

[nb_distribution.constant_median](#), [nb_distribution.constant_mean](#)
[nb_distribution.constant_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **constant_mean** ↑

`x = nb_distribution.constant_mean(m)`

Description:

Mean of a constant

Input:

- `m` : The constant

Output:

- `x` : The mean; `m`

See also:

`nb_distribution.constant_mode`, `nb_distribution.constant_median`
`nb_distribution.constant_variance`

Written by Kenneth Sæterhagen Paulsen

► **constant_median** ↑

`x = nb_distribution.constant_median(m)`

Description:

Median of a constant

Input:

- `m` : The constant

Output:

- `x` : The median; `m`

See also:

`nb_distribution.constant_mode`, `nb_distribution.constant_mean`
`nb_distribution.constant_variance`

Written by Kenneth Sæterhagen Paulsen

► **constant_mode** ↑

`x = nb_distribution.constant_mode(m)`

Description:

Mode of a constant

Input:

- `m` : The constant

Output:

- `x` : The mode; `m`

See also:

`nb_distribution.constant_median`, `nb_distribution.constant_mean`

`nb_distribution.constant_variance`

Written by Kenneth Sæterhagen Paulsen

► `constant_pdf` ↑

```
f = nb_distribution.constant_pdf(x,m)
```

Description:

PDF of a constant

Input:

- `x` : The point to evaluate the pdf, as a double
- `m` : The constant

Output:

- `f` : The PDF at the evaluated points, same size as `x`.

See also:

`nb_distribution.constant_cdf`, `nb_distribution.constant_rand`
`nb_distribution.constant_icdf`

Written by Kenneth Sæterhagen Paulsen

► `constant_rand` ↑

```
draws = nb_distribution.constant_rand(nrow,ncol,m)
```

Description:

"Draw" random numbers from distribution which always return the same number `m`. Same as `repmat(m,nrow,ncol)`.

Input:

- `nrow` : Number of rows of the matrix of the constant
- `ncol` : Number of columns of the matrix of the constant
- `m` : The constant

Output:

- draws : A nrow x ncol matrix of the number m

See also:

[nb_distribution.constant_pdf](#), [nb_distribution.constant_cdf](#)

Modified by Kenneth S. Paulsen

► **constant_skewness ↑**

x = nb_distribution.constant_skewness(m)

Description:

Skewness of a constant

Input:

- m : The constant

Output:

- x : 0

See also:

[nb_distribution.constant_median](#), [nb_distribution.constant_mean](#)
[nb_distribution.constant_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **constant_std ↑**

x = nb_distribution.constant_std(m)

Description:

Standard deviation of a constant

Input:

- m : The constant

Output:

- x : 0

See also:

[nb_distribution.constant_mode](#), [nb_distribution.constant_median](#)
[nb_distribution.constant_mean](#), [nb_distribution.constant_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **constant_variance** ↑

`x = nb_distribution.constant_variance(m)`

Description:

Variance of a constant

Input:

- `m` : The constant

Output:

- `x` : 0

See also:

[nb_distribution.constant_mode](#), [nb_distribution.constant_median](#)
[nb_distribution.constant_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **convert** ↑

`convert(obj)`

Description:

Convert a distribution to type 'kernel'

Input:

- `obj` : A matrix of nb_distributions objects

Output:

- `obj` : A matrix of nb_distributions objects with type set to 'kernel'

Written by Kenneth Sæterhagen Paulsen

► **defaultParameters** ↑

```
parameters = defaultParameters(type)
```

Description:

Get default parameters of a given distribution type.

Input:

- type : The type of distribution.

Output:

- parameters : A cell with the default parameters.

Written by Kenneth Sæterhagen Paulsen

► **domain** ↑

```
d = domain(obj)
```

Description:

Returns the full domain of the distribution represented by a nb_distribution object.

Input:

- obj : An object of class nb_distribution
- out : - 0 : Returns the lower and upper bound of the distribution, if truncated those limits are returned. Default.
- 1 : Returns the lower and upper bound of the distribution, if truncated those limits are not returned.
- 2 : Returns the lower and upper bound of the distribution, if truncated it will only return the lower truncation limit.
- 3 : Returns the lower and upper bound of the distribution, if truncated it will only return the upper truncation limit.

Output:

- domain : A nobjx2 double with the lower and upper limits of the domain.

Examples:

See also:

[nb_distribution.getEndOfDomain](#), [nb_distribution.getStartOfDomain](#)

Written by Kenneth Sæterhagen Paulsen

► **double2Dist** ↑

```
obj = nb_distribution.double2Dist(d)
```

Description:

Convert a double to a nb_distribution object representing a non-random variable, i.e. a constant.

Input:

- d : A double matrix. dim < 3

Output:

- obj : An object of class nb_distribution matching the size of d.

Examples:

```
obj = nb_distribution.double2Dist(rand(2,2))
```

Written by Kenneth Sæterhagen Paulsen

► **double2NormalDist** ↑

```
obj = nb_distribution.double2NormalDist(d,sigma)
```

Description:

Convert a double to a nb_distribution object representing a random univariate normal variable.

Input:

- d : A double matrix representing the mean. dim < 3

- sigma : A double matrix with same size as d representing the std.

Output:

```
- obj : An object of class nb_distribution matching the size of d.
```

Examples:

```
obj = nb_distribution.double2Dist(rand(2,2))
```

Written by Kenneth Sæterhagen Paulsen

► **empirical_cdf ↑**

```
f = nb_distribution.empirical_cdf(x,domain,density)
```

Description:

CDF of the (Kaplan-Meier) cumulative distribution function. To get the CDF of a point in between two points of the domain, a linear interpolation method is used.

Input:

```
- x : The points to evaluate the cdf, as a double (max dim is 3)  
- domain : The domain of the CDF.  
- CDF : The empirical cumulative density function.
```

Output:

```
- f : The CDF at the evaluated points, same size as x.
```

See also:

[nb_distribution.empirical_pdf](#), [nb_distribution.empirical_rand](#)
[nb_distribution.empirical_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **empirical_domain ↑**

```
f = nb_distribution.empirical_domain(domain)
```

Description:

Get the domain of the distribution

Input:

- domain : The domain of the distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **empirical_icdf** ↑

```
x = nb_distribution.empirical_icdf(p, domain, CDF)
```

Description:

Inverse CDF of the estimated empirical distribution. Uses the mean of the two closest points in the domain.

Input:

- p : A vector of probabilities
- domain : The domain of the distribution
- CDF : The cumulative density function

Output:

- x : The inverse CDF at the evaluated probabilities, same size as p.

See also:

[nb_distribution.empirical_pdf](#), [nb_distribution.empirical_rand](#)
[nb_distribution.empirical_cdf](#)

Written by Kenneth Sæterhagen Paulsen

► **empirical_mean** ↑

```
x = nb_distribution.empirical_mean(domain, density)
```

Description:

Mean of the estimated empirical density. This will use the [nb_distribution.empirical_rand](#) to make 1000 draws from the distribution, and then calculate the mean based on these draws.

Input:

- domain : The domain of the distribution
- CDF : The empirical CDF

Output:

- x : The mean of the estimated empirical density

See also:

[nb_distribution.empirical_median](#), [nb_distribution.empirical_mean](#)
[nb_distribution.empirical_variance](#), [nb_distribution.empirical_rand](#)

Written by Kenneth Sæterhagen Paulsen

► [**empirical_median**](#) ↑

x = nb_distribution.empirical_median(domain,density)

Description:

Median of the estimated empirical CDF. This will use the nb_distribution.empirical_icdf method to estimate the median.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The median of the estimated kernel density

See also:

[nb_distribution.kernel_mode](#), [nb_distribution.kernel_mean](#)
[nb_distribution.kernel_variance](#), [nb_distribution.kernel_rand](#)

Written by Kenneth Sæterhagen Paulsen

► [**empirical_mode**](#) ↑

x = nb_distribution.empirical_mode(domain,density)

Description:

The mode is found by finding the max point of the estimated PDF of the empirical CDF, and return the matching point in its domain.

Input:

- domain : The domain of the distribution.
- density : The cumulative density function of the distribution.

Output:

- x : The kurtosis of the estimated PDF of the empirical CDF density

See also:

[nb_distribution.empirical_median](#), [nb_distribution.empirical_mean](#)
[nb_distribution.empirical_variance](#), [nb_distribution.empirical_rand](#)

Written by Tobias Ingebrigtsen

► [empirical_pdf ↑](#)

```
f = nb_distribution.empirical_pdf(x,domain,CDF)
```

Description:

PDF of the estimated empirical density. To get the PDF of a point in between two points of the domain, a linear interpolation method is used.

Input:

- x : The points to evaluate the pdf, as a double.
- domain : The domain of the distribution.
- density : The empirical CDF.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.empirical_cdf](#), [nb_distribution.empirical_rand](#)
[nb_distribution.empirical_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [empirical_rand ↑](#)

```
draws = nb_distribution.empirical_rand(nrow,ncol,domain,CDF)
```

Description:

Draw random numbers from an estimated empirical density.

Input:

- nrow : Number of rows of the matrix drawn from the normal distribution.
- ncol : Number of columns of the matrix drawn from the normal distribution.
- domain : The domain of the distribution.
- CDF : The empirical CDF.

Output:

- draws : A nrow x ncol matrix of random numbers from the estimated empirical density.

See also:

[nb_distribution.empirical_pdf](#), [nb_distribution.empirical_cdf](#)

Written by Kenneth S. Paulsen

► **empirical_skewness ↑**

```
x = nb_distribution.empirical_skewness(domain,density)
```

Description:

Skewness of the estimated empirical density. This will use the `nb_distribution.empirical_rand` to make 1000 draws from the distribution, and then calculate the skewness based on these draws.

It will adjust the skewness for bias. See the `skewness` method by MATLAB for more on this

Input:

- domain : The domain of the distribution.
- density : The empirical CDF.

Output:

- x : The skewness of the estimated empirical density

See also:

`nb_distribution.empirical_median`, `nb_distribution.empirical_mean`
`nb_distribution.empirical_variance`, `nb_distribution.empirical_rand`

Written by Kenneth Sæterhagen Paulsen

► **`empirical_std`** ↑

`x = nb_distribution.empirical_std(domain, density)`

Description:

Standard deviation of the estimated empirical density. This will use the `nb_distribution.empirical_rand` to make 1000 draws from the distribution, and then calculate the standard deviation based on those draws.

It will adjust the standard deviation by T-1.

Input:

- `domain` : The domain of the distribution.
- `CDF` : The empirical CDF.

Output:

- `x` : The standard deviation of the estimated empirical density.

See also:

`nb_distribution.empirical_median`, `nb_distribution.empirical_mean`
`nb_distribution.empirical_variance`, `nb_distribution.empirical_rand`

Written by Kenneth Sæterhagen Paulsen

► **`empirical_variance`** ↑

`x = nb_distribution.empirical_variance(domain, CDF)`

Description:

Variance of the empirical density. This will use the `nb_distribution.empirical_rand` to make 1000 draws from the distribution, and then calculate the variance based on those draws.

It will adjust the variance by T-1.

Input:

- domain : The domain of the distribution.
- density : The empirical CDF.

Output:

- x : The variance of the estimated empirical density.

See also:

[nb_distribution.empirical_median](#), [nb_distribution.empirical_mean](#)
[nb_distribution.empirical_std](#), [nb_distribution.empirical_rand](#)

Written by Kenneth Sæterhagen Paulsen

► estimate ↑

```
obj = nb_distribution.estimate(x,xi,varargin)
```

Description:

Estimate a distribution using kernel density estimation.

Input:

- x : The assumed random observations of the distribution. As a nobs x 1 double.
- xi : The points where the distribution should be evaluated, i.e. the domain of the distribution. As a nobs x 1 double.
If not provided or given as empty the method `nb_distribution.estimateDomain` will be used.
- varargin : Optional input given to the `nb_ksdensity` function. Please see help for that function.

Output:

- obj : A `nb_distribution` object. See the property `domain` for the distributions `domain`, and `density` to see its PDF.

Examples:

```
x = randn(100,1);
obj = nb_distribution.estimate(x);
```

See also:

nb_distribution.estimateDomain

Written by Kenneth Sæterhagen Paulsen

► **estimateDomain** ↑

```
xi = nb_distribution.estimateDomain(x)
xi = nb_distribution.estimateDomain(x,numberOfStd,numDomain)
```

Description:

Estimate the domain of a distribution. Calculated as;

```
xi = min(x) - std(x):incr:max(x) + std(x)
```

Where;

```
incr = (xi(end) - xi(1))/999
```

Caution: The estimated domain will be estimated separately for each variable.

Input:

- x : The assumed random observation of the distribution.
As a nobs x nvars double.
- numberOfStd : Number of added space in the tails. In terms of the standard deviation. Default is 1.
- numDomain : Number of points of the returned domain. Default is 1000.

Output:

- xi : The domain of the distribution, as 1000 x nvars double

Written by Kenneth Sæterhagen Paulsen

► **exp_cdf** ↑

```
f = nb_distribution.exp_cdf(x,m)
```

Description:

CDF of the exponential distribution

Input:

- `x` : The point to evaluate the cdf, as a double
- `m` : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- `f` : The CDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.exp_pdf](#), [nb_distribution.exp_rand](#)
[nb_distribution.exp_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [exp_domain ↑](#)

`f = nb_distribution.exp_domain`

Description:

Get the domain of the exponential distribution

Output:

- `domain` : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► [exp_icdf ↑](#)

`x = nb_distribution.exp_icdf(p,m)`

Description:

Returns the inverse of the cdf at `p` of the exponential distribution

Input:

- `p` : A vector of probabilities
- `m` : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- `x` : A double with the quantile at each element of `p` of the $F(m, k)$ distribution

See also:

[nb_distribution.exp_cdf](#), [nb_distribution.exp_rand](#)
[nb_distribution.exp_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **exp_kurtosis** ↑

`x = nb_distribution.exp_kurtosis(m)`

Description:

Kurtosis of the exponential distribution.

Input:

- `m` : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- `x` : The kurtosis of the exponential distribution

See also:

[nb_distribution.exp_median](#), [nb_distribution.exp_mean](#)
[nb_distribution.exp_variance](#), [nb_distribution.exp_skewness](#)

Written by Kenneth Sæterhagen Paulsen

► **exp_mean** ↑

`x = nb_distribution.exp_mean(m)`

Description:

Mean of the exponential distribution.

Input:

- `m` : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- x : The mean of the exponential distribution

See also:

[nb_distribution.exp_mode](#), [nb_distribution.exp_mean](#)
[nb_distribution.exp_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **exp_median** ↑

x = nb_distribution.exp_median(m)

Description:

Median of the exponential distribution

Input:

- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- x : The median of the exponential distribution

See also:

[nb_distribution.exp_mode](#), [nb_distribution.exp_mean](#)
[nb_distribution.exp_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **exp_mode** ↑

x = nb_distribution.exp_mode(m)

Description:

Mode of the exponential distribution.

Input:

- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- `x` : The mode of the exponential distribution

See also:

`nb_distribution.exp_median`, `nb_distribution.exp_mean`
`nb_distribution.exp_variance`

Written by Kenneth Sæterhagen Paulsen

► **`exp_pdf`** ↑

```
f = nb_distribution.exp_pdf(x,m)
```

Description:

PDF of the exponential distribution

Input:

- `x` : The point to evaluate the pdf, as a double

- `m` : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- `f` : The PDF at the evaluated points, same size as `x`.

See also:

`nb_distribution.exp_cdf`, `nb_distribution.exp_rand`
`nb_distribution.exp_icdf`

Written by Kenneth Sæterhagen Paulsen

► **`exp_rand`** ↑

```
draws = nb_distribution.exp_rand(nrow,ncol,m)
```

Description:

Draw random numbers from the exponential distribution

Input:

- nrow : Number of rows of the matrix drawn from the exponential distribution
- ncol : Number of columns of the matrix drawn from the exponential distribution
- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- draws : A nrow x ncol matrix of random numbers from the exponential distribution

See also:

[nb_distribution.exp_pdf](#), [nb_distribution.exp_cdf](#)

Modified by Kenneth S. Paulsen

► [**exp_skewness ↑**](#)

`x = nb_distribution.exp_skewness(m)`

Description:

Skewness of the exponential distribution.

Input:

- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- x : The skewness of the exponential distribution

See also:

[nb_distribution.exp_median](#), [nb_distribution.exp_mean](#)
[nb_distribution.exp_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [**exp_std ↑**](#)

`x = nb_distribution.exp_std(m)`

Description:

Standard deviation of the exponential distribution.

Input:

- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- x : The standard deviation of the exponential distribution

See also:

[nb_distribution.exp_mode](#), [nb_distribution.exp_median](#)
[nb_distribution.exp_mean](#), [nb_distribution.exp_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **exp_variance ↑**

x = nb_distribution.exp_variance(m)

Description:

Variance of the exponential distribution.

Input:

- m : The rate parameter of the distribution. Must be positive 1x1 double.

Output:

- x : The variance of the exponential distribution

See also:

[nb_distribution.exp_mode](#), [nb_distribution.exp_median](#)
[nb_distribution.exp_mean](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **f_cdf ↑**

f = nb_distribution.f_cdf(x,m,k)

Description:

CDF of the F(m, k) distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- f : The CDF at the evaluated points, same size as x .

See also:

[nb_distribution.f_pdf](#), [nb_distribution.f_rand](#)
[nb_distribution.f_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [f_domain ↑](#)

`f = nb_distribution.f_domain`

Description:

Get the domain of the F-distribution

Output:

- $domain$: A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► [f_icdf ↑](#)

`x = nb_distribution.f_icdf(p, m, k)`

Description:

Returns the inverse of the cdf at p of the F(m, k) distribution

Input:

- p : A vector of probabilities
- m : First parameter of the distribution. Must be positive.
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : A double with the quantile at each element of p of the $F(m, k)$ distribution

See also:

[nb_distribution.f_cdf](#), [nb_distribution.f_rand](#)
[nb_distribution.f_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► f_kurtosis ↑

`x = nb_distribution.f_kurtosis(m, k)`

Description:

Kurtosis of the $F(m, k)$ distribution. Only defined for $k > 8$, otherwise not defined (will return nan).

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The kurtosis of the $F(m, k)$ distribution

See also:

[nb_distribution.f_median](#), [nb_distribution.f_mean](#)
[nb_distribution.f_variance](#), [nb_distribution.f_skewness](#)

Written by Kenneth Sæterhagen Paulsen

► **f_mean** ↑

```
x = nb_distribution.f_mean(m, k)
```

Description:

Mean of the $F(m, k)$ distribution. Only defined for $k > 2$, otherwise not defined (will return nan).

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The mean of the $F(m, k)$ distribution

See also:

[nb_distribution.f_mode](#), [nb_distribution.f_median](#)
[nb_distribution.f_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **f_median** ↑

```
x = nb_distribution.f_median(m, k)
```

Description:

Median of the $F(m, k)$ distribution

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The median of the $F(m, k)$ distribution

See also:

[nb_distribution.f_mode](#), [nb_distribution.f_mean](#)
[nb_distribution.f_variance](#)

Written by Kenneth Å!terhagen Paulsen

► **f_mode** ↑

`x = nb_distribution.f_mode(m, k)`

Description:

Mode of the $F(m, k)$ distribution. Only defined for $m > 2$, otherwise not defined (will return nan).

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The mode of the $F(m, k)$ distribution

See also:

[nb_distribution.f_median](#), [nb_distribution.f_mean](#)
[nb_distribution.f_variance](#)

Written by Kenneth Å!terhagen Paulsen

► **f_pdf** ↑

`f = nb_distribution.f_pdf(x, m, k)`

Description:

PDF of the $F(m, k)$ distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : First parameter of the distribution. Must be positive.
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- f : The PDF at the evaluated points, same size as x .

See also:

[nb_distribution.f_cdf](#), [nb_distribution.f_rand](#)
[nb_distribution.f_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **f_rand** ↑

```
draws = nb_distribution.f_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the $F(m, k)$ distribution

Input:

- $nrow$: Number of rows of the matrix drawn from the $F(m, k)$ distribution
- $ncol$: Number of columns of the matrix drawn from the $F(m, k)$ distribution
- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- $draws$: A $nrow \times ncol$ matrix of random numbers from the $F(m, k)$ distribution

See also:

[nb_distribution.f_pdf](#), [nb_distribution.f_cdf](#)

Modified by Kenneth S. Paulsen

► **f_skewness** ↑

```
x = nb_distribution.f_skewness(m,k)
```

Description:

Skewness of the $F(m, k)$ distribution. Only defined for $k > 6$. Will return nan otherwise.

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The skewness of the $F(m, k)$ distribution

See also:

[nb_distribution.f_median](#), [nb_distribution.f_mean](#)
[nb_distribution.f_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **f_std** ↑

```
x = nb_distribution.f_std(m,k)
```

Description:

Standard deviation of the $F(m, k)$ distribution. Only defined for $k > 4$. Will return nan otherwise.

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The standard deviation of the $F(m, k)$ distribution

See also:

[nb_distribution.f_mode](#), [nb_distribution.f_median](#)
[nb_distribution.f_mean](#), [nb_distribution.f_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **f_variance ↑**

```
x = nb_distribution.f_variance(m,k)
```

Description:

Variance of the F(m, k) distribution. Only defined for $k > 4$. Will return nan otherwise.

Input:

- m : First parameter of the distribution. Must be positive
- k : Second parameter of the distribution. A parameter such that the mean of the F-distribution is equal to $k/(k-2)$ for $k > 2$. Must be positive.

Output:

- x : The variance of the F(m, k) distribution

See also:

[nb_distribution.f_mode](#), [nb_distribution.f_median](#)
[nb_distribution.f_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **fgamma_cdf ↑**

```
f = nb_distribution.fgamma_cdf(x,m,k)
```

Description:

CDF of the flipped gamma distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : A parameter such that the mean of the flipped gamma = $-m*k$
- k : A parameter such that the variance of the flipped gamma = $m*(k^2)$

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.fgamma_pdf](#), [nb_distribution.fgamma_rand](#)
[nb_distribution.fgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [fgamma_domain](#) ↑

f = nb_distribution.fgamma_domain

Description:

Get the domain of the flipped gamma distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► [fgamma_icdf](#) ↑

x = nb_distribution.fgamma_icdf(p,m,k)

Description:

Returns the inverse of the cdf at p of the flipped gamma(m,k) distribution.

Input:

- p : A vector of probabilities

- m : A parameter such that the mean of the flipped gamma = -m*k

- k : A parameter such that the variance of the flipped gamma = m*(k^2)

Output:

- x : A double with the quantile at each element of p of the flipped gamma(m,k) distribution

See also:

[nb_distribution.fgamma_cdf](#), [nb_distribution.fgamma_rand](#)
[nb_distribution.fgamma_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **fgamma_kurtosis** ↑

`x = nb_distribution.fgamma_kurtosis(m,k)`

Description:

Kurtosis of the flipped gamma distribution

Input:

- `m` : A parameter such that the mean of the flipped gamma = $-m*k$
- `k` : A parameter such that the variance of the flipped gamma = $m*(k^2)$

Output:

- `x` : The kurtosis of the flipped gamma distribution

See also:

[nb_distribution.fgamma_median](#), [nb_distribution.fgamma_mean](#)
[nb_distribution.fgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **fgamma_mean** ↑

`x = nb_distribution.fgamma_mean(m,k)`

Description:

Mean of the flipped gamma distribution

Input:

- `m` : A parameter such that the mean of the flipped gamma = $-m*k$
- `k` : A parameter such that the variance of the flipped gamma = $m*(k^2)$

Output:

- `x` : The mean of the flipped gamma distribution

See also:

`nb_distribution.fgamma_mode`, `nb_distribution.fgamma_median`
`nb_distribution.fgamma_variance`

Written by Kenneth SÃ¶terhagen Paulsen

► **`fgamma_median`** ↑

`x = nb_distribution.fgamma_median(m, k)`

Description:

Median of the flipped gamma distribution

Input:

- `m` : A parameter such that the mean of the flipped gamma = $-m*k$
- `k` : A parameter such that the variance of the flipped gamma = $m*(k^2)$

Output:

- `x` : The median of the flipped gamma distribution

See also:

`nb_distribution.fgamma_mode`, `nb_distribution.fgamma_mean`
`nb_distribution.fgamma_variance`

Written by Kenneth SÃ¶terhagen Paulsen

► **`fgamma_mode`** ↑

`x = nb_distribution.fgamma_mode(m, k)`

Description:

Mode of the flipped gamma distribution

Input:

- `m` : A parameter such that the mean of the flipped gamma = $-m*k$
- `k` : A parameter such that the variance of the flipped gamma = $m*(k^2)$

Output:

- x : The mode of the flipped gamma distribution

See also:

[nb_distribution.fgamma_median](#), [nb_distribution.fgamma_mean](#)
[nb_distribution.fgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [fgamma_pdf](#) ↑

f = nb_distribution.fgamma_pdf(x,m,k)

Description:

PDF of the flipped gamma distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : A parameter such that the mean of the flipped gamma = -m*k
- k : A parameter such that the variance of the flipped gamma = m*(k^2)

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.fgamma_cdf](#), [nb_distribution.fgamma_rand](#)
[nb_distribution.fgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [fgamma_rand](#) ↑

draws = nb_distribution.fgamma_rand(nrow,ncol,m,k)

Description:

Draw random numbers from the flipped gamma distribution

Input:

```
- nrow : Number of rows of the matrix drawn from the gamma distribution  
- ncol : Number of columns of the matrix drawn from the gamma distribution  
- m : A parameter such that the mean of the flipped gamma = -m*k  
- k : A parameter such that the variance of the flipped gamma = m*(k^2)
```

Output:

```
- draws : A nrow x ncol matrix of random numbers from the flipped gamma distribution
```

See also:

[nb_distribution.fgamma_pdf](#), [nb_distribution.fgamma_cdf](#)

Modified by Kenneth S. Paulsen

► **fgamma_skewness ↑**

```
x = nb_distribution.fgamma_skewness(m, k)
```

Description:

Skewness of the flipped gamma distribution

Input:

```
- m : A parameter such that the mean of the flipped gamma = -m*k  
- k : A parameter such that the variance of the flipped gamma = m*(k^2)
```

Output:

```
- x : The skewness of the flipped gamma distribution
```

See also:

[nb_distribution.fgamma_median](#), [nb_distribution.fgamma_mean](#)
[nb_distribution.fgamma_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **fgamma_std ↑**

```
x = nb_distribution.fgamma_std(m, k)
```

Description:

Standard deviation of the flipped gamma distribution

Input:

- m : A parameter such that the mean of the flipped gamma = -m*k
- k : A parameter such that the variance of the flipped gamma = m*(k^2)

Output:

- x : The standard deviation of the flipped gamma distribution

See also:

[nb_distribution.fgamma_mode](#), [nb_distribution.fgamma_median](#)
[nb_distribution.fgamma_mean](#), [nb_distribution.fgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **fgamma_variance ↑**

```
x = nb_distribution.fgamma_variance(m, k)
```

Description:

Variance of the flipped gamma distribution

Input:

- m : A parameter such that the mean of the flipped gamma = -m*k
- k : A parameter such that the variance of the flipped gamma = m*(k^2)

Output:

- x : The variance of the flipped gamma distribution

See also:

[nb_distribution.fgamma_mode](#), [nb_distribution.fgamma_median](#)
[nb_distribution.fgamma_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **[finvgamma_cdf](#)** ↑

```
f = nb_distribution.finvgamma_cdf(x,m,k)
```

Description:

CDF of the flipped invgamma distribution

Input:

- *x* : The point to evaluate the cdf, as a double
- *m* : The shape parameter, as a double >0.
- *k* : The scale parameter, as a double >0.

Output:

- *f* : The CDF at the evaluated points, same size as *x*.

See also:

[nb_distribution.finvgamma_pdf](#), [nb_distribution.finvgamma_rand](#)
[nb_distribution.finvgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **[finvgamma_domain](#)** ↑

```
f = nb_distribution.finvgamma_domain
```

Description:

Get the domain of the flipped invgamma distribution

Output:

- *domain* : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **[finvgamma_icdf](#)** ↑

```
x = nb_distribution.finvgamma_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the flipped invgamma(m,k) distribution

Input:

- p : A vector of probabilities
- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : A double with the quantile at each element of p of the flipped invgamma(m,k) distribution

See also:

[nb_distribution.finvgamma_cdf](#), [nb_distribution.finvgamma_rand](#)
[nb_distribution.finvgamma_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_kurtosis ↑**

x = nb_distribution.finvgamma_kurtosis(m,k)

Description:

Kurtosis of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The kurtosis of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_median](#), [nb_distribution.finvgamma_mean](#)
[nb_distribution.finvgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_mean** ↑

```
x = nb_distribution.finvgamma_mean(m,k)
```

Description:

Mean of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The mean of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_mode](#), [nb_distribution.finvgamma_median](#)
[nb_distribution.finvgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_median** ↑

```
x = nb_distribution.finvgamma_median(m,k)
```

Description:

Median of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The median of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_mode](#), [nb_distribution.finvgamma_mean](#)
[nb_distribution.finvgamma_variance](#)

► **finvgamma_mode** ↑

```
x = nb_distribution.finvgamma_mode(m,k)
```

Description:

Mode of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The mode of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_median](#), [nb_distribution.finvgamma_mean](#)
[nb_distribution.finvgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_pdf** ↑

```
f = nb_distribution.finvgamma_pdf(x,m,k)
```

Description:

PDF of the flipped invgamma distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.finvgamma_cdf](#), [nb_distribution.finvgamma_rand](#)
[nb_distribution.finvgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_rand** ↑

```
draws = nb_distribution.finvgamma_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the flipped invgamma distribution

Input:

- nrow : Number of rows of the matrix drawn from the invgamma distribution
- ncol : Number of columns of the matrix drawn from the invgamma distribution
- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- draws : A nrow x ncol matrix of random numbers from the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_pdf](#), [nb_distribution.finvgamma_cdf](#)

Modified by Kenneth S. Paulsen

► **finvgamma_skewness** ↑

```
x = nb_distribution.finvgamma_skewness(m, k)
```

Description:

Skewness of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.

- k : The scale parameter, as a double >0.

Output:

- x : The skewness of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_median](#), [nb_distribution.finvgamma_mean](#)
[nb_distribution.finvgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_std ↑**

x = nb_distribution.finvgamma_std(m, k)

Description:

Standard deviation of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.

- k : The scale parameter, as a double >0.

Output:

- x : The standard deviation of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_mode](#), [nb_distribution.finvgamma_median](#)
[nb_distribution.finvgamma_mean](#), [nb_distribution.finvgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **finvgamma_variance ↑**

x = nb_distribution.finvgamma_variance(m, k)

Description:

Variance of the flipped invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The variance of the flipped invgamma distribution

See also:

[nb_distribution.finvgamma_mode](#), [nb_distribution.finvgamma_median](#)
[nb_distribution.finvgamma_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_cdf** ↑

f = nb_distribution.gamma_cdf(x, m, k)

Description:

CDF of the gamma distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : A parameter such that the mean of the gamma = m*k
- k : A parameter such that the variance of the gamma = m*(k^2)

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.gamma_pdf](#), [nb_distribution.gamma_rand](#)
[nb_distribution.gamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_domain** ↑

f = nb_distribution.gamma_domain

Description:

Get the domain of the gamma distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth SÃ¶terhagen Paulsen

► **gamma_icdf** ↑

```
x = nb_distribution.gamma_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the gamma(m,k) distribution

Input:

- p : A vector of probabilities
- m : A parameter such that the mean of the gamma = m*k
- k : A parameter such that the variance of the gamma = m*(k^2)

Output:

- x : A double with the quantile at each element of p of the gamma(m,k) distribution

See also:

[nb_distribution.gamma_cdf](#), [nb_distribution.gamma_rand](#)
[nb_distribution.gamma_pdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **gamma_kurtosis** ↑

```
x = nb_distribution.gamma_kurtosis(m,k)
```

Description:

Kurtosis of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- x : The kurtosis of the gamma distribution

See also:

[nb_distribution.gamma_median](#), [nb_distribution.gamma_mean](#)
[nb_distribution.gamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_mean** ↑

x = nb_distribution.gamma_mean(m, k)

Description:

Mean of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- x : The mean of the gamma distribution

See also:

[nb_distribution.gamma_mode](#), [nb_distribution.gamma_median](#)
[nb_distribution.gamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_median** ↑

x = nb_distribution.gamma_median(m, k)

Description:

Median of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = m*k
- k : A parameter such that the variance of the gamma = m*(k^2)

Output:

- x : The median of the gamma distribution

See also:

[nb_distribution.gamma_mode](#), [nb_distribution.gamma_mean](#)
[nb_distribution.gamma_variance](#)

Written by Kenneth SÃ¶derhagen Paulsen

► **gamma_mode** ↑

x = nb_distribution.gamma_mode(m,k)

Description:

Mode of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = m*k
- k : A parameter such that the variance of the gamma = m*(k^2)

Output:

- x : The mode of the gamma distribution

See also:

[nb_distribution.gamma_median](#), [nb_distribution.gamma_mean](#)
[nb_distribution.gamma_variance](#)

Written by Kenneth SÃ¶derhagen Paulsen

► **gamma_pdf** ↑

f = nb_distribution.gamma_pdf(x,m,k)

Description:

PDF of the gamma distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- f : The PDF at the evaluated points, same size as x .

See also:

[nb_distribution.gamma_cdf](#), [nb_distribution.gamma_rand](#)
[nb_distribution.gamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_rand** ↑

draws = nb_distribution.gamma_rand(nrow, ncol, m, k)

Description:

Draw random numbers from the Gamma distribution

Input:

- $nrow$: Number of rows of the matrix drawn from the gamma distribution
- $ncol$: Number of columns of the matrix drawn from the gamma distribution
- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- $draws$: A $nrow \times ncol$ matrix of random numbers from the gamma distribution

See also:

[nb_distribution.gamma_pdf](#), [nb_distribution.gamma_cdf](#)

Modified by Kenneth S. Paulsen

► **gamma_skewness** ↑

```
x = nb_distribution.gamma_skewness(m,k)
```

Description:

Skewness of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- x : The skewness of the gamma distribution

See also:

[nb_distribution.gamma_median](#), [nb_distribution.gamma_mean](#)
[nb_distribution.gamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **gamma_std** ↑

```
x = nb_distribution.gamma_std(m,k)
```

Description:

Standard deviation of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m \cdot k$
- k : A parameter such that the variance of the gamma = $m \cdot (k^2)$

Output:

- x : The standard deviation of the gamma distribution

See also:

[nb_distribution.gamma_mode](#), [nb_distribution.gamma_median](#)
[nb_distribution.gamma_mean](#), [nb_distribution.gamma_variance](#)

► **gamma_variance** ↑

```
x = nb_distribution.gamma_variance(m,k)
```

Description:

Variance of the gamma distribution

Input:

- m : A parameter such that the mean of the gamma = m*k
- k : A parameter such that the variance of the gamma = m*(k^2)

Output:

- x : The variance of the gamma distribution

See also:

[nb_distribution.gamma_mode](#), [nb_distribution.gamma_median](#)
[nb_distribution.gamma_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **get** ↑

```
propertyValue = get(obj,propertyName)
```

Description:

Get a property value of a nb_distribution object.

Type properties(obj) in the command line to get a list of supported properties, and type help nb_distribution.<propertyName> to get help on a specific property.

Input:

- obj : An object of class nb_distribution
- propertyName : A string with the property name

Output:

- `propertyValue` : The return value of the wanted property.

Examples:

```
propertyValue = get(obj,'type');
```

Written by Kenneth S. Paulsen

► **getEndOfDomain** ↑

```
start = getEndOfDomain(obj)
```

Description:

Get the approximate end point of the domain of all the distributions.
Uses the 99th percentile.

Input:

- `obj` : A vector of `nb_distribution` objects

Output:

- `finish` : The start point of the distribution, as `1x1 double`

See also:

[nb_distribution.domain](#)

Written by Kenneth Sæterhagen Paulsen

► **getNumberOfParameters** ↑

```
value = getNumberOfParameters(obj)
```

Description:

Get the number of parameters of the distribution.

Output:

- `value` : The number of parameters of the object.

Written by Kenneth Sæterhagen Paulsen

► **getParameters** ↑

```
value = getParameters(obj)
```

Description:

Get the parameter values.

Output:

- value : The parameters of the object, as 1 x nParam double.

Written by Kenneth Sæterhagen Paulsen

► **getStartOfDomain** ↑

```
start = getStartOfDomain(obj)
```

Description:

Get the approximate start point of the domain of all the distributions.
Uses the 1st percentile.

Input:

- obj : A vector of nb_distribution objects

Output:

- start : The start point of the distribution, as 1x1 double

See also:

[nb_distribution.domain](#)

Written by Kenneth Sæterhagen Paulsen

► **getTheoreticalHistCounts** ↑

```
hist = getTheoreticalHistCounts(obj,intervals)
```

Description:

Get the theoretical hist counts of the distribution. The output is per observation, so you need to multiply it by the number of observation to get the final histogram counts.

Input:

- obj : A nb_distribution object.
- intervals : A 1 x N double with upper bound of the intervals of the return histogram. intervals(end), should be the upper bound of the domain of the distribution.

Output:

- hist : A 1 x N double with the theoretical count of each interval.

Written by Kenneth Sæterhagen Paulsen

► hist_cdf ↑

```
f = nb_distribution.hist_cdf(x,a,b,c,d,e)
```

Description:

"CDF" type histogram.

Input:

- x : The bin points of the histogram, as a vector double.
- a : The data points, as a nobs x 1 double.

Output:

- f : The "CDF" at the evaluated bins, same size as x.

See also:

[nb_distribution.hist_pdf](#), [nb_distribution.hist_rand](#)
[nb_distribution.hist_icdf](#), [nb_mci](#)

Written by Kenneth Sæterhagen Paulsen

► hist_domain ↑

```
f = nb_distribution.hist_domain(a)
```

Description:

Get the domain of the histogram, i.e. min and max of the empirical data.

Input:

- a : The data points.

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **hist_icdf** ↑

```
x = nb_distribution.hist_icdf(p,a,b,c,d,e)
```

Description:

Inverse "CDF" type histogram.

Input:

- p : A vector of probabilities
- a : The data points, as a nobs x 1 double.

Output:

- x : A double with the quantile at each element of p of the gamma(m,k) distribution

See also:

[nb_distribution.hist_cdf](#), [nb_distribution.hist_rand](#)
[nb_distribution.hist_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_kurtosis** ↑

```
x = nb_distribution.hist_kurtosis(a,b,c,d,e)
```

Description:

Empirical kurtosis.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The kurtosis of the data.

See also:

[nb_distribution.hist_median](#), [nb_distribution.hist_mean](#)
[nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_mean** ↑

```
x = nb_distribution.hist_mean(a)
```

Description:

Empirical mean.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The mean of the data.

See also:

[nb_distribution.hist_mode](#), [nb_distribution.hist_median](#)
[nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_median** ↑

```
x = nb_distribution.hist_median(a)
```

Description:

Empirical median.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The median of the data.

See also:

[nb_distribution.hist_mode](#), [nb_distribution.hist_mean](#)
[nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_mode** ↑

x = nb_distribution.hist_mode(a)

Description:

Empirical mode. Calculated using a kernel method.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The mode of the data.

See also:

[nb_distribution.hist_median](#), [nb_distribution.hist_mean](#)
[nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_pdf** ↑

f = nb_distribution.hist_pdf(x,a)

Description:

"PDF" type histogram.

Input:

- x : The bin points of the histogram, as a vector double

- a : The data points, as a nobs x 1 double.

Output:

- f : The "PDF" at the evaluated points, same size as x.

See also:

[nb_distribution.hist_cdf](#), [nb_distribution.hist_rand](#)
[nb_distribution.hist_icdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **hist_rand** ↑

draws = nb_distribution.hist_rand(nrow, ncol, a)

Description:

This function is not supported!

Input:

- nrow : Number of rows of the matrix drawn from the distribution
- ncol : Number of columns of the matrix drawn from the distribution
- a : The data points, as a nobs x 1 double.

Output:

- draws : No output returned.

See also:

[nb_distribution.hist_pdf](#), [nb_distribution.hist_cdf](#)

Modified by Kenneth S. Paulsen

► **hist_skewness** ↑

x = nb_distribution.hist_skewness(a)

Description:

Empirical skewness.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The skewness of the data.

See also:

[nb_distribution.hist_median](#), [nb_distribution.hist_mean](#)
[nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_std** ↑

x = nb_distribution.hist_std(a)

Description:

Empirical standard deviation.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- x : The standard deviation of the data.

See also:

[nb_distribution.hist_mode](#), [nb_distribution.hist_median](#)
[nb_distribution.hist_mean](#), [nb_distribution.hist_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **hist_variance** ↑

x = nb_distribution.hist_variance(a)

Description:

Empirical variance.

Input:

- a : The data points, as a nobs x 1 double.

Output:

- `x` : The variance of the data.

See also:

`nb_distribution.hist_mode`, `nb_distribution.hist_median`
`nb_distribution.hist_mean`

Written by Kenneth Sæterhagen Paulsen

► **horzcat** ↑

`obj = horzcat(varargin)`

Description:

Horizontal concatenation

Input:

- `varargin` : Either an object of class `nb_distribution` or an object of class `double`

Output:

- `obj` : An object of class `nb_distribution`

Written by Kenneth Sæterhagen Paulsen

► **icdf** ↑

`f = icdf(obj,x)`

Description:

Evaluate the inverse cdf of the given distribution at the value(s) `x`.

Input:

- `obj` : An object of class `nb_distribution`

- `x` : A double with any size if `numel(obj) == 1`, otherwise it must be a `Tx1` double.

Output:

- `f` : `numel(obj) == 1` : A double with the same size as `x`
otherwise : A double with size `T x nobj`

Written by Kenneth Sæterhagen Paulsen

► **increment** ↑

```
increment(obj,type,index,value,smoothing)
```

Description:

Increment the probability density at a selected point with a given value. Please note that the underlying object must be of type 'kernel'.

Input:

- obj : An object of class nb_distribution.
- typeOfDensity : Either 'pdf' or 'cdf'.
- index : The point in the domain of the distribution to increment. See the property parameters.
- value : The value to increment the selected point in the domain. Must be lower than a certain value, as the density at any point cannot be higher than 1.
- smoothing :> An integer : The number of elements to smmoth out in the neighbourhood of the selected point.
 - > 'kernel' : Using a kernel smoother on the random draws produced by the incremented distribution. This will not reproduce the selected points density, as much of its density will be smoothed out to the neighbourhood.
 - > [] : No smoothing.

See also:

[nb_distribution.decrement](#), [nb_distribution.convert](#)

Written by Kenneth SÅ!terhagen Paulsen

► **initialize** ↑

```
obj = nb_distribution.initialize(varargin)
```

Description:

Intialize a vector of nb_distribution objects with 'propertyName', 'propertyValue' pairs. The propertyValue part must be a cell with same size for all properties set, i.e. with the same size as the number of created distributions.

This is a static method

Input:

- varargin : See the description of the method

Output:

- obj : A vector of nb_distribution objects

Examples:

```
obj = nb_distribution.initialize('type',{'normal','gamma'},...
    'parameters',{{6,2},{2,2}});
plot(obj,2:0.1:16,'pdf')
```

See also:

Written by Kenneth Sæterhagen Paulsen

► **invgamma_cdf** ↑

```
f = nb_distribution.invgamma_cdf(x,m,k)
```

Description:

CDF of the invgamma distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.invgamma_pdf](#), [nb_distribution.invgamma_rand](#)
[nb_distribution.invgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **invgamma_domain** ↑

```
f = nb_distribution.invgamma_domain
```

Description:

Get the domain of the invgamma distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth SÃ¶terhagen Paulsen

► **invgamma_icdf ↑**

```
x = nb_distribution.invgamma_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the invgamma(m,k) distribution

Input:

- p : A vector of probabilities
- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : A double with the quantile at each element of p of the invgamma(m,k) distribution

See also:

[nb_distribution.invgamma_cdf](#), [nb_distribution.invgamma_rand](#)
[nb_distribution.invgamma_pdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **invgamma_kurtosis ↑**

```
x = nb_distribution.invgamma_kurtosis(m,k)
```

Description:

Kurtosis of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The kurtosis of the invgamma distribution

See also:

[nb_distribution.invgamma_median](#), [nb_distribution.invgamma_mean](#)
[nb_distribution.invgamma_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► [invgamma_mean ↑](#)

x = nb_distribution.invgamma_mean(m, k)

Description:

Mean of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The mean of the invgamma distribution

See also:

[nb_distribution.invgamma_mode](#), [nb_distribution.invgamma_median](#)
[nb_distribution.invgamma_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► [invgamma_median ↑](#)

x = nb_distribution.invgamma_median(m, k)

Description:

Median of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The median of the invgamma distribution

See also:

[nb_distribution.invgamma_mode](#), [nb_distribution.invgamma_mean](#)
[nb_distribution.invgamma_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► [invgamma_mode](#) ↑

x = nb_distribution.invgamma_mode(m, k)

Description:

Mode of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The mode of the invgamma distribution

See also:

[nb_distribution.invgamma_median](#), [nb_distribution.invgamma_mean](#)
[nb_distribution.invgamma_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► [invgamma_pdf](#) ↑

f = nb_distribution.invgamma_pdf(x, m, k)

Description:

PDF of the invgamma distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The shape parameter, as a double >0 .
- k : The scale parameter, as a double >0 .

Output:

- f : The PDF at the evaluated points, same size as x .

See also:

[nb_distribution.invgamma_cdf](#), [nb_distribution.invgamma_rand](#)
[nb_distribution.invgamma_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [invgamma_rand](#) ↑

```
draws = nb_distribution.invgamma_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the invgamma distribution

Input:

- $nrow$: Number of rows of the matrix drawn from the invgamma distribution
- $ncol$: Number of columns of the matrix drawn from the invgamma distribution
- m : The shape parameter, as a double >0 .
- k : The scale parameter, as a double >0 .

Output:

- $draws$: A $nrow \times ncol$ matrix of random numbers from the invgamma distribution

See also:

[nb_distribution.invgamma_pdf](#), [nb_distribution.invgamma_cdf](#)

► **invgamma_skewness** ↑

```
x = nb_distribution.invgamma_skewness(m,k)
```

Description:

Skewness of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The skewness of the invgamma distribution

See also:

[nb_distribution.invgamma_median](#), [nb_distribution.invgamma_mean](#)
[nb_distribution.invgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **invgamma_std** ↑

```
x = nb_distribution.invgamma_std(m,k)
```

Description:

Standard deviation of the invgamma distribution

Input:

- m : The shape parameter, as a double >0.
- k : The scale parameter, as a double >0.

Output:

- x : The standard deviation of the invgamma distribution

See also:

`nb_distribution.invgamma_mode`, `nb_distribution.invgamma_median`
`nb_distribution.invgamma_mean`, `nb_distribution.invgamma_variance`

Written by Kenneth Sæterhagen Paulsen

► **invgamma_variance** ↑

```
x = nb_distribution.invgamma_variance(m,k)
```

Description:

Variance of the invgamma distribution

Input:

- `m` : The shape parameter, as a double >0.
- `k` : The scale parameter, as a double >0.

Output:

- `x` : The variance of the invgamma distribution

See also:

`nb_distribution.invgamma_mode`, `nb_distribution.invgamma_median`
`nb_distribution.invgamma_mean`

Written by Kenneth Sæterhagen Paulsen

► **invwish_rand** ↑

```
A = invwish_rand(h,n)
A = invwish_rand(h,n,type)
```

Description:

Draws an $m \times m$ matrix from a inverse wishart distribution with scale matrix `h` and degrees of freedom `nu = n`. This procedure uses Bartlett's decomposition.

Note: Parameterized so that mean is `h/n`

Input:

- `h` : $m \times m$ scale matrix.
- `n` : scalar degrees of freedom.
- `type` : 'cov' or 'covrepair'. See `nb_chol`

Output:

- A : m x m matrix draw from the inverse wishart distribution.

Written by Kenneth Sæterhagen Paulsen

► isnan ↑

Description:

Input:**Output:****Examples:****See also:**

Written by Kenneth Sæterhagen Paulsen

► kernel_cdf ↑

```
f = nb_distribution.kernel_cdf(x,domain,density)
```

Description:

CDF of the estimated kernel density. To get the CDF of a point in between two points of the domain, a linear interpolation method is used.

Input:

- x : The points to evaluate the cdf, as a double (max dim is 3)
- domain : The domain of the distribution
- density : The density of the distribution

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.kernel_pdf](#), [nb_distribution.kernel_rand](#)
[nb_distribution.kernel_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **kernel_domain** ↑

```
f = nb_distribution.kernel_domain(domain)
```

Description:

Get the domain of the distribution

Input:

- domain : The domain of the distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **kernel_icdf** ↑

```
x = nb_distribution.kernel_icdf(p, domain, density)
```

Description:

Inverse CDF of the estimated kernel distribution. Uses the mean of the two closest points in the domain.

Input:

- p : A vector of probabilities
- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The inverse CDF at the evaluated probabilities, same size as p.

See also:

[nb_distribution.kernel_pdf](#), [nb_distribution.kernel_rand](#)
[nb_distribution.kernel_cdf](#)

Written by Kenneth Sæterhagen Paulsen

► **kernel_kurtosis** ↑

```
x = nb_distribution.kernel_kurtosis(domain,density)
```

Description:

Kurtosis of the estimated kernel density. This will use the nb_distribution.kernel_rand make 1000 draws from the distribution, and then calculate the kurtois based on these draws.

It will adjust the kurtosis for bias. See the kurtosis method by MATLAB for more on this

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The kurtosis of the estimated kernel density

See also:

[nb_distribution.normal_median](#), [nb_distribution.normal_mean](#)
[nb_distribution.normal_variance](#), [nb_distribution.kernel_rand](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **kernel_mean** ↑

```
x = nb_distribution.kernel_mean(domain,density)
```

Description:

Mean of the estimated kernel density. This will use the nb_distribution.kernel_rand to make 1000 draws from the distribution, and then calculate the mean based on these draws.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The mean of the estimated kernel density

See also:

[nb_distribution.kernel_median](#), [nb_distribution.kernel_mean](#)
[nb_distribution.kernel_variance](#), [nb_distribution.kernel_rand](#)

Written by Kenneth Sæterhagen Paulsen

► [kernel_median ↑](#)

x = nb_distribution.kernel_median(domain,density)

Description:

Median of the estimated kernel density. This will use the nb_distribution.kernel_icdf method to estimate the median.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The median of the estimated kernel density

See also:

[nb_distribution.kernel_mode](#), [nb_distribution.kernel_mean](#)
[nb_distribution.kernel_variance](#), [nb_distribution.kernel_rand](#)

Written by Kenneth Sæterhagen Paulsen

► [kernel_mode ↑](#)

x = nb_distribution.kernel_mode(domain,density)

Description:

The mode is found by finding the max point of the estimated kernel density, and return the matching point in its domain.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The kurtosis of the estimated kernel density

See also:

[nb_distribution.kernel_median](#), [nb_distribution.kernel_mean](#)
[nb_distribution.kernel_variance](#), [nb_distribution.kernel_rand](#)

Written by Kenneth Sæterhagen Paulsen

► **kernel_pdf** ↑

```
f = nb_distribution.kernel_pdf(x,domain,density)
```

Description:

PDF of the estimated kernel density. To get the PDF of a point in between two points of the domain, a linear interpolation method is used.

Input:

- x : The points to evaluate the pdf, as a double
- domain : The domain of the distribution
- density : The density of the distribution

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.kernel_cdf](#), [nb_distribution.kernel_rand](#)
[nb_distribution.kernel_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **kernel_rand** ↑

```
draws = nb_distribution.kernel_rand(nrow,ncol,domain,density)
```

Description:

Draw random numbers from an estimated kernel density.

Input:

- nrow : Number of rows of the matrix drawn from the normal distribution
- ncol : Number of columns of the matrix drawn from the normal distribution
- domain : The domain of the distribution
- density : The density of the distribution

Output:

- draws : A nrow x ncol matrix of random numbers from the estimated kernel density

See also:

[nb_distribution.kernel_pdf](#), [nb_distribution.kernel_cdf](#)

Written by Kenneth S. Paulsen

► **kernel_skewness ↑**

`x = nb_distribution.kernel_skewness(domain,density)`

Description:

Skewness of the estimated kernel density. This will use the `nb_distribution.kernel_rand` to make 1000 draws from the distribution, and then calculate the skewness based on these draws.

It will adjust the skewness for bias. See the `skewness` method by MATLAB for more on this

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The skewness of the estimated kernel density

See also:

```
nb_distribution.kernel_median, nb_distribution.kernel_mean  
nb_distribution.kernel_variance, nb_distribution.kernel_rand
```

Written by Kenneth SÃ¶terhagen Paulsen

► **kernel_std** ↑

```
x = nb_distribution.kernel_std(domain,density)
```

Description:

Standard deviation of the estimated kernel density. This will use the nb_distribution.kernel_rand to make 1000 draws from the distribution, and then calculate the standard deviation based on those draws.

It will adjust the standard deviation by T-1.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The standard deviation of the estimated kernel density

See also:

```
nb_distribution.kernel_median, nb_distribution.kernel_mean  
nb_distribution.kernel_variance, nb_distribution.kernel_rand
```

Written by Kenneth SÃ¶terhagen Paulsen

► **kernel_variance** ↑

```
x = nb_distribution.kernel_variance(domain,density)
```

Description:

Variance of the estimated kernel density. This will use the nb_distribution.kernel_rand to make 1000 draws from the distribution, and then calculate the variance based on those draws.

It will adjust the variance by T-1.

Input:

- domain : The domain of the distribution
- density : The density of the distribution

Output:

- x : The variance of the estimated kernel density

See also:

[nb_distribution.kernel_median](#), [nb_distribution.kernel_mean](#)
[nb_distribution.kernel_std](#), [nb_distribution.kernel_rand](#)

Written by Kenneth Sæterhagen Paulsen

► **kurtosis** ↑

x = kurtosis(obj)

Description:

Evaluate the kurtosis of the given distribution.

The kurtosis will be adjusted for bias. See the function kurtosis made by MATLAB inc.

Input:

- obj : An object of class nb_distribution

Output:

- x : numel(obj) == 1 : A double with size as 1 x 1
- otherwise : A double with size 1 x nobj

Written by Kenneth Sæterhagen Paulsen

► **laplace_cdf** ↑

f = nb_distribution.laplace_cdf(x,m)

Description:

CDF of the laplace distribution

Input:

- `x` : The point to evaluate the cdf, as a double
- `m` : The location parameter. Median = `m`. Must be a number.
- `k` : Scale parameter. Must be a positive number.

Output:

- `f` : The CDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.laplace_pdf](#), [nb_distribution.laplace_rand](#)
[nb_distribution.laplace_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_domain** ↑

`f = nb_distribution.laplace_domain`

Description:

Get the domain of the laplace distribution

Output:

- `domain` : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **laplace_icdf** ↑

`x = nb_distribution.laplace_icdf(p,m,k)`

Description:

Returns the inverse of the cdf at `p` of the laplace distribution

Input:

- `p` : A vector of probabilities
- `m` : The location parameter. Median = `m`. Must be a number.
- `k` : Scale parameter. Must be a positive number.

Output:

- `x` : A double with the quantile at each element of `p` of the `laplace(m,k)` distribution

See also:

[nb_distribution.laplace_cdf](#), [nb_distribution.laplace_rand](#)
[nb_distribution.laplace_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► laplace_kurtosis ↑

`x = nb_distribution.laplace_kurtosis(m,k)`

Description:

Kurtosis of the laplace distribution

Input:

- `m` : The location parameter. Median = `m`. Must be a number.
- `k` : Scale parameter. Must be a positive number.

Output:

- `x` : The kurtosis of the laplace distribution

See also:

[nb_distribution.laplace_median](#), [nb_distribution.laplace_mean](#)
[nb_distribution.laplace_variance](#)

Written by Kenneth Sæterhagen Paulsen

► laplace_mean ↑

`x = nb_distribution.laplace_mean(m,k)`

Description:

Mean of the laplace distribution

Input:

- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mean of the laplace distribution

See also:

[nb_distribution.laplace_mode](#), [nb_distribution.laplace_median](#)
[nb_distribution.laplace_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_median** ↑

x = nb_distribution.laplace_median(m, k)

Description:

Median of the laplace distribution

Input:

- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The median of the laplace distribution

See also:

[nb_distribution.laplace_mode](#), [nb_distribution.laplace_mean](#)
[nb_distribution.laplace_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_mode** ↑

x = nb_distribution.laplace_mode(m, k)

Description:

Mode of the laplace distribution

Input:

- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mode of the laplace distribution

See also:

[nb_distribution.laplace_median](#), [nb_distribution.laplace_mean](#)
[nb_distribution.laplace_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_pdf** ↑

```
f = nb_distribution.laplace_pdf(x,m,k)
```

Description:

PDF of the laplace distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.laplace_cdf](#), [nb_distribution.laplace_rand](#)
[nb_distribution.laplace_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_rand** ↑

```
draws = nb_distribution.laplace_rand(nrow,ncol,m,k)
```

Description:

Draw random numbers from the laplace distribution

Input:

- nrow : Number of rows of the matrix drawn from the laplace distribution
- ncol : Number of columns of the matrix drawn from the laplace distribution
- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- draws : A nrow x ncol matrix of random numbers from the laplace distribution

See also:

[nb_distribution.laplace_pdf](#), [nb_distribution.laplace_cdf](#)

Modified by Kenneth S. Paulsen

► **laplace_skewness ↑**

`x = nb_distribution.laplace_skewness(m, k)`

Description:

Skewness of the laplace distribution

Input:

- m : The location parameter. Median = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The skewness of the laplace distribution

See also:

[nb_distribution.laplace_median](#), [nb_distribution.laplace_mean](#)
[nb_distribution.laplace_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **laplace_std** ↑

```
x = nb_distribution.laplace_std(m, k)
```

Description:

Standard deviation of the laplace distribution

Input:

- m : The location paramter. Mode = m. Must be a number.
- k : Scale paramter. Must be a positive number.

Output:

- x : The standard deviation of the laplace distribution

See also:

[nb_distribution.laplace_mode](#), [nb_distribution.laplace_median](#)
[nb_distribution.laplace_mean](#), [nb_distribution.laplace_variance](#)

Written by Kenneth Sætherhagen Paulsen

► **laplace_variance** ↑

```
x = nb_distribution.laplace_variance(m, k)
```

Description:

Variance of the laplace distribution

Input:

- m : The location paramter. Median = m. Must be a number.
- k : Scale paramter. Must be a positive number.

Output:

- x : The variance of the laplace distribution

See also:

[nb_distribution.laplace_mode](#), [nb_distribution.laplace_median](#)
[nb_distribution.laplace_mean](#)

Written by Kenneth Sæterhagen Paulsen

► logistic_cdf ↑

```
f = nb_distribution.logistic_cdf(x,m)
```

Description:

CDF of the logistic distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.logistic_pdf](#), [nb_distribution.logistic_rand](#)
[nb_distribution.logistic_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► logistic_domain ↑

```
f = nb_distribution.logistic_domain
```

Description:

Get the domain of the logistic distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► logistic_icdf ↑

```
x = nb_distribution.logistic_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the logistic distribution

Input:

- p : A vector of probabilities
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : A double with the quantile at each element of p of the logistic(m,k) distribution

See also:

[nb_distribution.logistic_cdf](#), [nb_distribution.logistic_rand](#)
[nb_distribution.logistic_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **logistic_kurtosis ↑**

```
x = nb_distribution.logistic_kurtosis(m,k)
```

Description:

Kurtosis of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The kurtosis of the logistic distribution

See also:

[nb_distribution.logistic_median](#), [nb_distribution.logistic_mean](#)
[nb_distribution.logistic_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **[logistic_mean](#)** ↑

```
x = nb_distribution.logistic_mean(m, k)
```

Description:

Mean of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mean of the logistic distribution

See also:

[nb_distribution.logistic_mode](#), [nb_distribution.logistic_median](#)
[nb_distribution.logistic_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **[logistic_median](#)** ↑

```
x = nb_distribution.logistic_median(m, k)
```

Description:

Median of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The median of the logistic distribution

See also:

[nb_distribution.logistic_mode](#), [nb_distribution.logistic_mean](#)
[nb_distribution.logistic_variance](#)

► **logistic_mode** ↑

```
x = nb_distribution.logistic_mode(m,k)
```

Description:

Mode of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The mode of the logistic distribution

See also:

[nb_distribution.logistic_median](#), [nb_distribution.logistic_mean](#)
[nb_distribution.logistic_variance](#)

► **logistic_pdf** ↑

```
f = nb_distribution.logistic_pdf(x,m,k)
```

Description:

PDF of the logistic distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.logistic_cdf](#), [nb_distribution.logistic_rand](#)
[nb_distribution.logistic_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **logistic_rand** ↑

```
draws = nb_distribution.logistic_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the logistic distribution

Input:

- nrow : Number of rows of the matrix drawn from the logistic distribution
- ncol : Number of columns of the matrix drawn from the logistic distribution
- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- draws : A nrow x ncol matrix of random numbers from the logistic distribution

See also:

[nb_distribution.logistic_pdf](#), [nb_distribution.logistic_cdf](#)

Modified by Kenneth S. Paulsen

► **logistic_skewness** ↑

```
x = nb_distribution.logistic_skewness(m, k)
```

Description:

Skewness of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The skewness of the logistic distribution

See also:

[nb_distribution.logistic_median](#), [nb_distribution.logistic_mean](#)
[nb_distribution.logistic_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **logistic_std ↑**

x = nb_distribution.logistic_std(m, k)

Description:

Standard deviation of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The standard deviation of the logistic distribution

See also:

[nb_distribution.logistic_mode](#), [nb_distribution.logistic_median](#)
[nb_distribution.logistic_mean](#), [nb_distribution.logistic_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **logistic_variance ↑**

x = nb_distribution.logistic_variance(m, k)

Description:

Variance of the logistic distribution

Input:

- m : The location parameter. Mode = m. Must be a number.
- k : Scale parameter. Must be a positive number.

Output:

- x : The variance of the logistic distribution

See also:

[nb_distribution.logistic_mode](#), [nb_distribution.logistic_median](#)
[nb_distribution.logistic_mean](#)

Written by Kenneth Sæterhagen Paulsen

► [lognormal_cdf ↑](#)

f = nb_distribution.lognormal_cdf(x, m, k)

Description:

CDF of the lognormal distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.lognormal_pdf](#), [nb_distribution.lognormal_rand](#)
[nb_distribution.lognormal_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [lognormal_domain ↑](#)

f = nb_distribution.lognormal_domain

Description:

Get the domain of the lognormal distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth SÃ¶terhagen Paulsen

► lognormal_icdf ↑

```
x = nb_distribution.lognormal_icdf(p,m,k)
```

Description:

Inverse CDF of the lognormal density

Input:

- p : A vector of probabilities
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The inverse CDF at the evaluated probabilities, same size as p.

See also:

[nb_distribution.lognormal_pdf](#), [nb_distribution.lognormal_rand](#)
[nb_distribution.lognormal_cdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► lognormal_kurtosis ↑

```
x = nb_distribution.lognormal_kurtosis(m,k)
```

Description:

Kurtosis of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The kurtosis of the lognormal distribution

See also:

[nb_distribution.lognormal_median](#), [nb_distribution.lognormal_mean](#)
[nb_distribution.lognormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_mean** ↑

x = nb_distribution.lognormal_mean(m, k)

Description:

Mean of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The mean of the lognormal distribution

See also:

[nb_distribution.lognormal_mode](#), [nb_distribution.lognormal_median](#)
[nb_distribution.lognormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_median** ↑

x = nb_distribution.lognormal_median(m, k)

Description:

Median of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$

Output:

- x : The median of the lognormal distribution

See also:

[nb_distribution.lognormal_mode](#), [nb_distribution.lognormal_mean](#)
[nb_distribution.lognormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_mode** ↑

x = nb_distribution.lognormal_mode(m, k)

Description:

Mode of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$

Output:

- x : The mode of the lognormal distribution

See also:

[nb_distribution.lognormal_median](#), [nb_distribution.lognormal_mean](#)
[nb_distribution.lognormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_pdf** ↑

f = nb_distribution.lognormal_pdf(x, m, k)

Description:

PDF of the lognormal distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- f : The PDF at the evaluated points, same size as x .

See also:

[nb_distribution.lognormal_cdf](#), [nb_distribution.lognormal_rand](#)
[nb_distribution.lognormal_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_rand** ↑

draws = nb_distribution.lognormal_rand(nrow, ncol, m, k)

Description:

Draw random numbers from the lognormal distribution

Input:

- $nrow$: Number of rows of the matrix drawn from the lognormal distribution
- $ncol$: Number of columns of the matrix drawn from the lognormal distribution
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- $draws$: A $nrow \times ncol$ matrix of random numbers from the lognormal distribution

See also:

[nb_distribution.lognormal_pdf](#), [nb_distribution.lognormal_cdf](#)

► **lognormal_skewness** ↑

```
x = nb_distribution.lognormal_skewness(m,k)
```

Description:

Skewness of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The skewness of the lognormal distribution

See also:

[nb_distribution.lognormal_median](#), [nb_distribution.lognormal_mean](#)
[nb_distribution.lognormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_std** ↑

```
x = nb_distribution.lognormal_std(m,k)
```

Description:

Standard deviation of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The standard deviation of the lognormal distribution

See also:

[nb_distribution.lognormal_mode](#), [nb_distribution.lognormal_median](#)
[nb_distribution.lognormal_mean](#), [nb_distribution.lognormal_std](#)

Written by Kenneth Sæterhagen Paulsen

► **lognormal_variance ↑**

```
x = nb_distribution.lognormal_variance(m, k)
```

Description:

Variance of the lognormal distribution

Input:

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The variance of the lognormal distribution

See also:

[nb_distribution.lognormal_mode](#), [nb_distribution.lognormal_median](#)
[nb_distribution.lognormal_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **mean ↑**

```
x = mean(obj)
```

Description:

Evaluate the mean of the given distribution.

Input:

- obj : An object of class nb_distribution

Output:

- x : $\text{numel}(\text{obj}) == 1$: A double with size as 1 x 1
- otherwise : A double with size nobj1 x nobj2

Written by Kenneth Sæterhagen Paulsen

► **meanshift_cdf** ↑

```
f = nb_distribution.meanshift_cdf(x,dist,param,lb,ub,ms)
```

Description:

CDF of the mean shifted possibly truncated distribution.

Input:

- `x` : The point to evaluate the cdf, as a double
- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.
- `ub` : Upper bound of the truncated distribution.
- `ms` : Mean shift parameter.

Output:

- `f` : The CDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.meanshift_pdf](#), [nb_distribution.meanshift_rand](#)
[nb_distribution.meanshift_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_domain** ↑

```
f = nb_distribution.meanshift_domain(dist,param,lb,ub,ms)
```

Description:

Get the domain of the mean shifted possibly truncated distribution.

Inputs:

- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.

- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **meanshift_icdf** ↑

```
x = nb_distribution.meanshift_icdf(p,dist,param,lb,ub,ms)
```

Description:

Returns the inverse of the CDF of the mean shifted possibly truncated distribution.

Input:

- p : A vector of probabilities
- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : A double with the quantile at each element of p of the truncated distribution

See also:

[nb_distribution.meanshift_cdf](#), [nb_distribution.meanshift_rand](#)
[nb_distribution.meanshift_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_kurtosis** ↑

```
x = nb_distribution.meanshift_kurtosis(dist,param,lb,ub,ms)
```

Description:

Kurtosis of the mean shifted and possibly truncated distribution. The calculation may be simulation based and may vary (if the distribution is truncated). Set seed to prevent this, or see the kurtosis method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : The kurtosis of the truncated distribution

See also:

[nb_distribution.meanshift_median](#), [nb_distribution.meanshift_mean](#)
[nb_distribution.meanshift_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_mean** ↑

```
x = nb_distribution.meanshift_mean(dist,param,lb,ub,ms)
```

Description:

Mean of the mean shifted and possibly truncated distribution. The calculation may be simulation based and may vary (if the distribution is truncated). Set seed to prevent this, or see the mean method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.

- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : The mean of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_mode](#), [nb_distribution.meanshift_median](#)
[nb_distribution.meanshift_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_median ↑**

x = nb_distribution.meanshift_median(dist,param,lb,ub,ms)

Description:

Median of the mean shifted and possibly truncated distribution.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : The median of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_mode](#), [nb_distribution.meanshift_mean](#)
[nb_distribution.meanshift_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_mode ↑**

```
x = nb_distribution.meanshift_mode(dist,param,lb,ub,ms)
```

Description:

Mode of the mean shifted and possibly truncated distribution.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : The mode of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_median](#), [nb_distribution.meanshift_mean](#)
[nb_distribution.meanshift_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_pdf ↑**

```
f = nb_distribution.meanshift_pdf(x,dist,param,lb,ub,ms)
```

Description:

PDF of the mean shifted possibly truncated distribution.

Input:

- x : The point to evaluate the pdf, as a double
- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.meanshift_cdf](#), [nb_distribution.meanshift_rand](#)
[nb_distribution.meanshift_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_rand** ↑

```
draws = nb_distribution.meanshift_rand(nrow,ncol,dist,param,lb,ub,ms)
```

Description:

Draw random number from a mean shifted possibly truncated distribution.

Input:

- nrow : Number of rows of the matrix drawn from the student-t distribution
- ncol : Number of columns of the matrix drawn from the student-t distribution
- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- draws : A nrow x ncol matrix of random numbers from the mean shifted possibly truncated distribution.

Written by Kenneth Sæterhagen Paulsen

► **meanshift_skewness** ↑

```
x = nb_distribution.meanshift_skewness(dist,param,lb,ub,ms)
```

Description:

Skewness of the mean shifted and possibly truncated distribution. The calculation may be simulation based and may vary (if the distribution is truncated). Set seed to prevent this, or see the skewness method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- x : The skewness of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_median](#), [nb_distribution.meanshift_mean](#)
[nb_distribution.meanshift_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **meanshift_std ↑**

`x = nb_distribution.meanshift_std(dist,param,lb,ub,ms)`

Description:

Standard deviation of the mean shifted and possibly truncated distribution.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.
- ms : Mean shift parameter.

Output:

- `x` : The standard deviation of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_mode](#), [nb_distribution.meanshift_median](#)
[nb_distribution.meanshift_mean](#), [nb_distribution.meanshift_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **meanshift_variance** ↑

`x = nb_distribution.meanshift_variance(dist,param,lb,ub,ms)`

Description:

Variance of the mean shifted and possibly truncated distribution.
The calculation may be simulation based and may vary (if the distribution is truncated). Set `seed` to prevent this, or see the `variance` method of the `nb_distribution` class.

Input:

- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.
- `ub` : Upper bound of the truncated distribution.
- `ms` : Mean shift parameter.

Output:

- `x` : The variance of the mean shifted and possibly truncated distribution.

See also:

[nb_distribution.meanshift_mode](#), [nb_distribution.meanshift_median](#)
[nb_distribution.meanshift_mean](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **median** ↑

```
x = median(obj)
```

Description:

Evaluate the median of the given distribution.

Input:

- obj : An object of class nb_distribution

Output:

- x : numel(obj) == 1 : A double with size as 1 x 1
- otherwise : A double with size 1 x nobj

Written by Kenneth Sæterhagen Paulsen

► **mle** ↑

```
obj = nb_distribution.mle(x,distribution)
```

Description:

Estimate a distribution using maximum likelihood estimator.

Input:

- x : The assumed random observation of the distribution. As a nobs x nvars double.
- dist : A string with the distribution to match to the data.

The supported distributions are:

```
> 'ast'  
> 'beta'  
> 'chis'  
> 'constant'  
> 'exp'  
> 'f'  
> 'gamma'  
> 'invgamma'  
> 'laplace'  
> 'logistic'  
> 'lognormal'  
> 'normal'  
> 't'  
> 'uniform'  
> 'wald'
```

Type <help nb_distribution.type> to get a description of the of the different distributions.

Output:

- obj : A 1 x nvars nb_distribution object array.

Examples:

```
x = randn(100,1);
obj = nb_distribution.mle(x,'normal');
```

See also:

[nb_distribution.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► [mme](#) ↑

```
[obj,params] = nb_distribution.mme(x,dist,lb,ub)
```

Description:

Estimate a distribution using metods of moment estimator.

Input:

- x : The assumed random observation of the distribution. As a nobs x 1 double.

- dist : A string with the distribution to match to the data.

The supported distributions are:

```
> 'beta'
> 'chis'
> 'constant'
> 'exp'
> 'f'
> 'gamma'
> 'invgamma'
> 'laplace'
> 'logistic'
> 'lognormal'
> 'normal'
> 'skt'
> 't'
> 'tri'
> 'uniform'
> 'wald'
```

Type `<help nb_distribution.type>` to get a description of the of the different distributions.

Output:

```
- obj      : A nb_distribution object.  
- params : The hyperparameters as a cell array
```

Examples:

```
x = randn(100,1);  
obj = nb_distribution.mme(x,'normal');
```

See also:

[nb_distribution.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **mode** ↑

```
x = mode(obj)
```

Description:

Evaluate the mode of the given distribution.

Input:

```
- obj : An object of class nb_distribution
```

Output:

```
- x   : numel(obj) == 1 : A double with size as 1 x 1  
       otherwise        : A double with size 1 x nobj
```

Written by Kenneth Sæterhagen Paulsen

► **normal_cdf** ↑

```
f = nb_distribution.normal_cdf(x,m,k)
```

Description:

CDF of the normal distribution

Input:

```
- x : The point to evaluate the cdf, as a double  
- m : The mean of the distribution  
- k : The std of the distribution
```

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.normal_pdf](#), [nb_distribution.normal_rand](#)
[nb_distribution.normal_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **normal_domain** ↑

f = nb_distribution.normal_domain

Description:

Get the domain of the normal distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **normal_icdf** ↑

x = nb_distribution.normal_icdf(p,m,k)

Description:

Inverse CDF of the normal density

Input:

- p : A vector of probabilities
- m : The mean of the distribution
- k : The std of the distribution

Output:

- x : The inverse CDF at the evaluated probabilities, same size as p.

See also:

[nb_distribution.normal_pdf](#), [nb_distribution.normal_rand](#)

`nb_distribution.normal_cdf`

Written by Kenneth Sæterhagen Paulsen

► **normal_kurtosis ↑**

`x = nb_distribution.normal_kurtosis(m,k)`

Description:

Kurtosis of the normal distribution

Input:

- `m` : A parameter such that the mean of the normal is `m`
- `k` : A parameter such that the std of the normal is `k`

Output:

- `x` : The kurtosis of the normal distribution

See also:

`nb_distribution.normal_median`, `nb_distribution.normal_mean`
`nb_distribution.normal_variance`

Written by Kenneth Sæterhagen Paulsen

► **normal_mean ↑**

`x = nb_distribution.normal_mean(m,k)`

Description:

Mean of the normal distribution

Input:

- `m` : A parameter such that the mean of the normal is `m`
- `k` : A parameter such that the std of the normal is `k`

Output:

- `x` : The mean of the normal distribution

See also:

[nb_distribution.normal_mode](#), [nb_distribution.normal_median](#)
[nb_distribution.normal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [normal_median ↑](#)

`x = nb_distribution.normal_median(m, k)`

Description:

Median of the normal distribution

Input:

- `m` : A parameter such that the mean of the normal is `m`
- `k` : A parameter such that the std of the normal is `k`

Output:

- `x` : The median of the normal distribution

See also:

[nb_distribution.normal_mode](#), [nb_distribution.normal_mean](#)
[nb_distribution.normal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [normal_mode ↑](#)

`x = nb_distribution.normal_mode(m, k)`

Description:

Mode of the normal distribution

Input:

- `m` : A parameter such that the mean of the normal is `m`
- `k` : A parameter such that the std of the normal is `k`

Output:

- `x` : The mode of the normal distribution

See also:

[nb_distribution.normal_median](#), [nb_distribution.normal_mean](#)
[nb_distribution.normal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **normal_pdf ↑**

`f = nb_distribution.normal_pdf(x,m,k)`

Description:

PDF of the normal distribution

Input:

- `x` : The point to evaluate the cdf, as a double
- `m` : The mean of the distribution
- `k` : The std of the distribution

Output:

- `f` : The PDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.normal_cdf](#), [nb_distribution.normal_rand](#)
[nb_distribution.normal_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **normal_rand ↑**

`draws = nb_distribution.normal_rand(nrow,ncol,m,k)`

Description:

Draw random numbers from the normal distribution

Input:

```
- nrow : Number of rows of the matrix drawn from the normal distribution  
- ncol : Number of columns of the matrix drawn from the normal distribution  
- m : The mean of the distribution  
- k : The std of the distribution
```

Output:

```
- draws : A nrow x ncol matrix of random numbers from the normal distribution
```

See also:

[nb_distribution.normal_pdf](#), [nb_distribution.normal_cdf](#)

Written by Kenneth S. Paulsen

► **normal_skewness ↑**

```
x = nb_distribution.normal_skewness(m,k)
```

Description:

Skewness of the normal distribution

Input:

```
- m : A parameter such that the mean of the normal is m  
- k : A parameter such that the std of the normal is k
```

Output:

```
- x : The skewness of the normal distribution
```

See also:

[nb_distribution.normal_median](#), [nb_distribution.normal_mean](#)
[nb_distribution.normal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **normal_std ↑**

```
x = nb_distribution.normal_std(m,k)
```

Description:

Standard deviation of the normal distribution

Input:

- m : A parameter such that the mean of the normal is m
- k : A parameter such that the std of the normal is k

Output:

- x : The standard deviation of the normal distribution

See also:

[nb_distribution.normal_mode](#), [nb_distribution.normal_median](#)
[nb_distribution.normal_mean](#), [nb_distribution.normal_std](#)

Written by Kenneth Sæterhagen Paulsen

► **normal_variance ↑**

x = nb_distribution.normal_variance(m, k)

Description:

Variance of the normal distribution

Input:

- m : A parameter such that the mean of the normal = m
- k : A parameter such that the std of the normal = k

Output:

- x : The variance of the normal distribution

See also:

[nb_distribution.normal_mean](#), [nb_distribution.normal_std](#)

Written by Kenneth Sæterhagen Paulsen

► **parameterization ↑**

```
obj = nb_distribution.parameterization(m,v,dist)
obj = nb_distribution.parameterization(m,v,dist,lb,ub)
obj = nb_distribution.parameterization(m,v,dist,lb,ub,s,k,mo)
```

Description:

Back out the hyperparameters given some moments.

Supported distribution:

```
> Mean only: 'chis', 'constant', 'exp'
> Variance only: 't' (1 parameter family)
> Mean and variance only: 'beta', 'f', 'gamma', 'invgamma', 'laplace',
  'logistic', 'lognormal', 'normal', 'uniform', 'wald'
> Mean, variance and kurtosis: 't' (3 parameter family)
> Mean, variance, skewness and kurtosis: 'skt'
> Mean, mode and variance: 'tri'
> Mode and variance (Set mean (me input) to []): 'gamma'
```

Input:

- me : Mean of the distribution.
- v : Variance of the distribution.
- dist : A string with the distribution to match to the data.

See description for the supported distributions.

- lb : Lower bound of the distribution. Default is no lower bound.
- ub : Upper bound of the distribution. Default is no upper bound.
- s : Skewness of the distribution.
- k : Kurtosis of the distribution.
- mo : Mode of the distribution.

Output:

- obj : A nb_distribution object.
- params : The hyperparameters as a cell array

Examples:

```
x          = randn(100,1);
obj        = nb_distribution.parameterization(x,'normal');
[~, params] = nb_distribution.parameterization(x,'normal');
```

See also:

[nb_distribution.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **parametrization** ↑

```
obj = nb_distribution.parametrization(m,v,distribution)
obj = nb_distribution.parametrization(m,v,distribution,lb,ub)
obj = nb_distribution.parametrization(m,v,distribution,lb,ub,s,k,mo)
```

Description:

Deprecated method. See `nb_distribution.parameterization` instead.

See also:

[nb_distribution.parameterization](#)

Written by Kenneth Sæterhagen Paulsen

► **pdf** ↑

```
f = pdf(obj,x)
```

Description:

Evaluate the pdf of the given distribution at the value(s) `x`.

Input:

- `obj` : An object of class `nb_distribution`
- `x` : A double with any size if `numel(obj) == 1`, otherwise it must be a `Tx1` double.

Output:

- `f` : `numel(obj) == 1` : A double with the same size as `x`
otherwise : A double with size `T x nobj`

Written by Kenneth Sæterhagen Paulsen

► **perc2Dist** ↑

```
obj = nb_distribution.perc2Dist(perc,values,nDraws,gridpoints,iter, ...
                                varargin)
```

Description:

This method makes draws from the percentiles, and do a kernel density estimation based on those draws.

Be aware that there is no unique solution to this problem, as the estimated kernel density is only one of possible many distribution with the wanted percentiles!

Input:

- perc : The percentiles to fit. Must at least provide 3 percentiles. E.g. [10,30,50,70,90]
- values : The values of the distribution at the given percentiles.
- nDraws : Number of draws.
- gridpoints : The number of grid points of the domain. Default is 1000.
- iter : The number of iteration of the algorithm. Default is 1.
- varargin : Optional input given to the nb_ksdensity function. Please see help for that function.

Output:

- obj : An object of class nb_distribution

Examples:

```
obj = nb_distribution.perc2Dist([10,30,50,70,90],[1,4,6,7,8],1000);
plot(obj)
```

See also:

[nb_distribution.perc2DistCDF](#)

Written by Kenneth Sæterhagen Paulsen

► **perc2DistCDF** ↑

```
obj = nb_distribution.perc2DistCDF(perc,values,startD,endD, ...
gridpoints,nDraws,varargin)
```

Description:

This method makes draws from the percentiles, and do a kernel density estimation based on those draws.

Be aware that there is no unique solution to this problem, as the estimated kernel density is only one of possible many distribution with the wanted percentiles!

Input:

- perc : The percentiles to fit. Must at least provide 3 percentiles. E.g. [10,30,50,70,90]
- values : The values of the distribution at the given percentiles.
- startD : Start value of the domain. Will be the "0" percentile.
- endD : End value of the domain. Will be the "100" percentile.
- gridpoints : The number of grid points of the domain. Default is 1000.
- nDraws : The number of draws from the interpolated cdf to base the kernel density estimation on. Default is 1000.
- varargin : Optional input given to the nb_ksdensity function. Please see help for that function.

Output:

- obj : An object of class nb_distribution

Examples:

```
obj = nb_distribution.perc2DistCDF([10,30,50,70,90],[1,4,6,7,8],-1,14);
plot(obj)
```

See also:

[nb_distribution.perc2Dist](#)

Written by Kenneth Sæterhagen Paulsen

► perc2ParamDist ↑

```
[obj,err] = nb_distribution.perc2ParamDist(dist,perc,values,varargin)
```

Description:

This method estimate the parameters of the wanted distribution based on a set of percentiles. It will minimize the distance between the parameterized distribution and the provided percentiles.

Input:

- dist : A string with the distribution to match to the percentiles.
See help on the nb_distribution.type property for the supported distributions.
- perc : The percentiles to fit. Must at least provide as many

percentiles as there are parameters of the distribution.
E.g. [10,30,50,70,90]

- values : The values of the distribution at the given percentiles.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : An object of class nb_distribution
- err : Ask for this output if you want to return a potential error instead of it being thrown by this method. If an error occurs this output is non-empty, while obj is.

Examples:

```
orig = nb_distribution('type','normal','parameters',{0,2});
perc = [10,30,50,70,90];
values = percentile(orig,perc);
obj = nb_distribution.perc2ParamDist('normal',perc,values);
plot([orig,obj])
```

See also:

[nb_distribution.perc2Dist](#), [nb_callOptimizer](#)

Written by Kenneth Sæterhagen Paulsen

► **percentile ↑**

x = percentile(obj,perc)

Description:

Get the wanted percentiles of the distribution(s). Same as icdf.

Input:

- obj : A vector of nb_distribution objects
- perc : 1 x nPerc double with the wanted percentiles. E.g. [10,90]

Output:

```
- x      : A nPerc x nobj double
```

Examples:

```
obj(2) = nb_distribution('type','normal','parameters',{0,2})  
x      = percentile(obj,[10,90])
```

See also:

[nb_distribution.icdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **plot** ↑

```
plotter = plot(obj,type)
```

Description:

Graph either the PDF or CDF of the distribution

Input:

- obj : A nb_distribution object.
- x : The values to evaluate the distribution. x must be a Nx1 double.
If not provided or empty the full domain of the distribution(s) will be used
- type : A string with either 'pdf' (default) or 'cdf'

Output:

- plotter : A nb_graph_data object. If nargout is 0 the graph will be plotted automatically, otherwise you need to call the graph method on the plotter object.

Examples:

```
obj = nb_distribution  
plot(obj)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **plot3d** ↑

```
plotter = plot(obj,type)
```

Description:

Graph either the PDF or CDF of the distribution

Input:

- obj : A vector of nb_distribution objects.
- x : The values to evaluate the distribution. x must be a Nx1 double.
If not provided or empty the full domain of the distribution(s) will be used
- type : A string with either 'pdf' (default) or 'cdf'
- plot : To set plottype. Either 'surf' (default) or 'mesh'. Either as a string or a function handle.

Output:

- plotter : A nb_graph_data object. If nargout is 0 the graph will be plotted automatically, otherwise you need to call the graph method on the plotter object.

Examples:

Written by Kenneth Sæterhagen Paulsen

► qestimation ↑

```
dist = nb_distribution.qestimation(type, quantiles, values)
```

Description:

Estimate a distribution given known quantiles.

Caution : Only for the distribution family with two parameters, except for 'ast'.

Caution : This function minimizes the absolute deviation between the quantiles provided and the quantiles of the (parameterized) distribution.

Input:

- type : The type of distribution to estimate.
See the nb_distribution class for the supported distributions.
- quantiles : The 1xN quantiles. Should be between 0 and 1. N must be greater than or equal to the number of parameters (K) of

the distribution to estimate. If the distribution does not match perfectly a sum(abs(qDist-quantiles)) measure will be minimized. Here qDist is the quantiles produced by the provided distribution.

- values : The 1xN known values at the quantiles.

Optional input:

- 'optimizer' : 'lsqnonlin', 'fminsearch' or 'fmincon'.
- 'optimset' : A struct. See optimset function. If not given the default settings are used.
- 'init' : Initial values for the parameters. Depends on the type input:
 - > 'skt' : 1 x 4 double. See help on 4 last input to nb_distribution.skt_icdf
 - > 'ast' : 1 x 5 double. See help on 5 last input to nb_distribution.ast_icdf
 - > otherwise : 1 x 2 double. See help on 2 last input to nb_distribution.XXX_icdf, where XX needs to be substituted by type (name of distribution).

Output:

- dist : The found distribution as an nb_distribution object.

Examples:

```
dist1 = nb_distribution.qestimation('normal',[0.5,0.7],[0,0.5])
dist2 = nb_distribution.qestimation('normal',[0.5,0.7,0.9],[0,0.5,0.8])
dist3 = nb_distribution.qestimation('ast',[0.1,0.3,0.5,0.7],...
[0,0.5,0.7,0.9])
```

Written by Kenneth Sæterhagen Paulsen

► **random ↑**

```
draws = random(obj,nrow,ncol)
```

Description:

Draw random numbers from the distribution represented by the nb_distribution object.

Input:

- obj : An object of class nb_distribution
- nrow : The number of rows of the draws output
- ncol : The number of columns of the draws output

Output:

```
- draws : numel(obj) == 1 : A nrow x ncol double with the draws from the
                           distribution
otherwise           : A nrow x ncol x nobj1 x nobj2 double with the
                           draws from the distributions
```

Examples:

```
obj = nb_distribution;
d   = random(obj,10,1);
```

Written by Kenneth Sæterhagen Paulsen

► set ↑

```
set(obj,varargin)
```

Description:

Sets the properties of the nb_distribution object.

Type properties(obj) in the command line to get a list of supported properties, and type help nb_distribution.<propertyName> to get help on a specific property.

Caution : When the property type is set, the paramters property will be set to its default value given the type of the distribution. See the paramters for more on this.

Caution : When the property type is set, the name property will be set to its default value given the type and parameterization of the distribution.

Input:

```
- obj      : An object of class nb_distribution
- varargin : 'propertyName',propertyValue,...
```

Output:

```
- obj      : The input object with the updated properties.
```

Examples:

```
set(obj,'propertyName',propertyValue);
obj.set('propertyName',propertyValue,...)
obj = obj.set('type','gamma');
```

Written by Kenneth S. Paulsen

► **sim2KernelDist** ↑

```
dist = nb_distribution.sim2KernelDist(y)
dist = nb_distribution.sim2KernelDist(y,lb,ub,varargin)
```

Description:

Convert a simulated data to a nb_distribution object.

Input:

- y : A nHor x nVar x nDraws double matrix.
- lb : A nHor x nVar double matrix with the lower bounds. Set to empty if no bounds on any elements (default). Set to -inf to not bound a given element.
- ub : A nHor x nVar double matrix with the upper bounds. Set to empty if no bounds on any elements (default). Set to inf to not bound a given element.

Optional input:

- varargin : See varargin in nb_distribution.estimate. 'support' is not supported. See the lb and ub inputs instead.

Output:

- obj : An nHor x nVar object of class nb_distribution.

Examples:

```
obj = nb_distribution.sim2KernelDist(rand(2,2,1000))
```

See also:

[nb_distribution.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_cdf** ↑

```
f = nb_distribution.skewedt_cdf(x,a,b,c,d)
```

Description:

CDF of the generalized skewed t-distribution.

Caution : Calculated using monte carlo integration. This may induce numerical problems with calculating the CDF of this distribution.

Input:

- *x* : The point to evaluate the cdf, as a double.
- *a* : The location parameter (mean).
- *b* : The scale parameter.
- *c* : The skewness parameter.
- *d* : The kurtosis parameter.

Output:

- *f* : The CDF at the evaluated points, same size as *x*.

See also:

[nb_distribution.skewedt_pdf](#), [nb_distribution.skewedt_rand](#)
[nb_distribution.skewedt_icdf](#), [nb_mci](#)

Written by Kenneth Sæterhagen Paulsen

► [skewedt_domain](#) ↑

f = `nb_distribution.skewedt_domain`

Description:

Get the domain of the generalized skewed t-distribution

Output:

- *domain* : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► [skewedt_icdf](#) ↑

x = `nb_distribution.skewedt_icdf(p,a,b,c,d)`

Description:

Returns the inverse of the cdf at *p* of the generalized skewed t-distribution.

Caution : There are may be some numerical problems with calculating the CDF of this distribution, so the results in the tails may wrong.

Input:

- p : A vector of probabilities
- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : A double with the quantile at each element of p of the gamma(m,k) distribution

See also:

[nb_distribution.skewedt_cdf](#), [nb_distribution.skewedt_rand](#)
[nb_distribution.skewedt_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_kurtosis ↑**

x = nb_distribution.skewedt_kurtosis(a,b,c,d)

Description:

Kurtosis of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The kurtosis of the distribution

See also:

[nb_distribution.skewedt_median](#), [nb_distribution.skewedt_mean](#)
[nb_distribution.skewedt_variance](#)

► **skewedt_mean** ↑

```
x = nb_distribution.skewedt_mean(a,b,c,d)
```

Description:

Mean of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The mean of the distribution

See also:

[nb_distribution.skewedt_mode](#), [nb_distribution.skewedt_median](#)
[nb_distribution.skewedt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_median** ↑

```
x = nb_distribution.skewedt_median(a,b,c,d)
```

Description:

Median of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The median of the distribution

See also:

[nb_distribution.skewedt_mode](#), [nb_distribution.skewedt_mean](#)
[nb_distribution.skewedt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_mode** ↑

`x = nb_distribution.skewedt_mode(a,b,c,d)`

Description:

Mode of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The mode of the distribution

See also:

[nb_distribution.skewedt_median](#), [nb_distribution.skewedt_mean](#)
[nb_distribution.skewedt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_pdf** ↑

`f = nb_distribution.skewedt_pdf(x,a,b,c,d)`

Description:

PDF of the generalized skewed t-distribution.

Input:

- x : The point to evaluate the pdf, as a double
- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.skewedt_cdf](#), [nb_distribution.skewedt_rand](#)
[nb_distribution.skewedt_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► [skewedt_rand](#) ↑

```
draws = nb_distribution.skewedt_rand(nrow, ncol, a, b, c, d)
```

Description:

Draw random numbers from the generalized skewed t-distribution.

Caution : This method use the inverse cdf function to produce draws.
As the CDF is calculated by MCI it is prone to errors, this
may also result in draws that are wrong!

Input:

- nrow : Number of rows of the matrix drawn from the distribution
- ncol : Number of columns of the matrix drawn from the distribution
- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- draws : A nrow x ncol matrix of random numbers from the distribution

See also:

[nb_distribution.skewedt_pdf](#), [nb_distribution.skewedt_cdf](#)

Modified by Kenneth S. Paulsen

► **skewedt_skewness ↑**

`x = nb_distribution.skewedt_skewness(a,b,c,d)`

Description:

Skewness of the generalized skewed t-distribution.

Input:

- `a` : The location parameter (mean).
- `b` : The scale parameter.
- `c` : The skewness parameter.
- `d` : The kurtosis parameter.

Output:

- `x` : The skewness of the distribution

See also:

[nb_distribution.skewedt_median](#), [nb_distribution.skewedt_mean](#)
[nb_distribution.skewedt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_std ↑**

`x = nb_distribution.skewedt_std(a,b,c,d)`

Description:

Standard deviation of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The standard deviation of the distribution

See also:

[nb_distribution.skewedt_mode](#), [nb_distribution.skewedt_median](#)
[nb_distribution.skewedt_mean](#), [nb_distribution.skewedt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skewedt_variance** ↑

x = nb_distribution.skewedt_variance(a,b,c,d)

Description:

Variance of the generalized skewed t-distribution.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The skewness parameter.
- d : The kurtosis parameter.

Output:

- x : The variance of the distribution

See also:

[nb_distribution.skewedt_mode](#), [nb_distribution.skewedt_median](#)
[nb_distribution.skewedt_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **skewness** ↑

```
x = skewness(obj)
```

Description:

Evaluate the skewness of the given distribution.

The skewness will be adjusted for bias. See the function skewness made by MATLAB inc.

Input:

- obj : An object of class nb_distribution
- type : Type of skewness measure
 - > 'normal' : $E[(X - \text{mean}(X))^3] / \text{var}(X)^{2/3}$
 - > 'pearson1' : $(\text{mean}(X) - \text{mode}(X)) / \sqrt{\text{var}(X)}$
 - > 'pearson2' : $3 * (\text{mean}(X) - \text{median}(X)) / \sqrt{\text{var}(X)}$
 - > 'bowley' : Set u to 3/4 in the max problem below.
 - > 'quantile' : $\max [icdf(u) + icdf(1-u) - 2\text{median}(X) / \dots$
 $icdf(u) + icdf(1-u)]$
 - > 'paulsen' : $\text{cdf}(\text{mean}(X)) - (1 - \text{cdf}(\text{mean}(X)))$

Output:

- x : $\text{numel}(\text{obj}) == 1$: A double with size as 1 x 1
otherwise : A double with size 1 x nobj

Written by Kenneth Sæterhagen Paulsen

► **skt_cdf ↑**

```
f = nb_distribution.skt_cdf(x,a,b,c,d)
```

Description:

CDF of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- x : The point to evaluate the cdf, as a double.
- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

`nb_distribution.skt_pdf`, `nb_distribution.skt_rand`
`nb_distribution.skt_icdf`, `nb_mci`

Written by Kenneth Sæterhagen Paulsen

► **`skt_domain`** ↑

`f = nb_distribution.skt_domain`

Description:

Get the domain of the Azzalini skewed t-distribution

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. *J.Roy.Statist.Soc. B* 65, 367-389.

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **`skt_icdf`** ↑

`x = nb_distribution.skt_icdf(p,a,b,c,d)`

Description:

Returns the inverse of the cdf at p of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. *J.Roy.Statist.Soc. B* 65, 367-389.

Input:

- p : A vector of probabilities
- a : The location parameter (mean).
- b : The scale parameter.

- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : A double with the quantile at each element of p of the gamma(m,k) distribution

See also:

[nb_distribution.skt_cdf](#), [nb_distribution.skt_rand](#)
[nb_distribution.skt_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **skt_kurtosis ↑**

x = nb_distribution.skt_kurtosis(a,b,c,d)

Description:

Kurtosis of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The kurtosis of the distribution

See also:

[nb_distribution.skt_median](#), [nb_distribution.skt_mean](#)
[nb_distribution.skt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skt_mean** ↑

```
x = nb_distribution.skt_mean(a,b,c,d)
```

Description:

Mean of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367–389.

Input:

- a : The location parameter (mean).
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The mean of the distribution

See also:

[nb_distribution.skt_mode](#), [nb_distribution.skt_median](#)
[nb_distribution.skt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skt_median** ↑

```
x = nb_distribution.skt_median(a,b,c,d)
```

Description:

Median of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367–389.

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The median of the distribution

See also:

[nb_distribution.sklt_mode](#), [nb_distribution.sklt_mean](#)
[nb_distribution.sklt_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **skt_mode ↑**

```
x = nb_distribution.sklt_mode(a,b,c,d)
```

Description:

Mode of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The mode of the distribution

See also:

[nb_distribution.sklt_median](#), [nb_distribution.sklt_mean](#)
[nb_distribution.sklt_variance](#)

► **skt_pdf** ↑

```
f = nb_distribution.skt_pdf(x,a,b,c,d)
```

Description:

PDF of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- x : The point to evaluate the pdf, as a double
- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.skt_cdf](#), [nb_distribution.skt_rand](#)
[nb_distribution.skt_icdf](#)

Written by Kenneth Sætherhagen Paulsen

► **skt_rand** ↑

```
draws = nb_distribution.skt_rand(nrow,ncol,a,b,c,d)
```

Description:

Draw random numbers from the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- nrow : Number of rows of the matrix drawn from the distribution
- ncol : Number of columns of the matrix drawn from the distribution
- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- draws : A nrow x ncol matrix of random numbers from the distribution

See also:

[nb_distribution.skt_pdf](#), [nb_distribution.skt_cdf](#)

Modified by Kenneth S. Paulsen

► skt_skewness ↑

`x = nb_distribution.skt_skewness(a,b,c,d)`

Description:

Skewness of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t-distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The skewness of the distribution

See also:

[nb_distribution.skt_median](#), [nb_distribution.skt_mean](#)
[nb_distribution.skt_variance](#)

Written by Kenneth Åström Paulsen

► **skt_std ↑**

`x = nb_distribution.skt_std(a,b,c,d)`

Description:

Standard deviation of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. *J.Roy.Statist.Soc. B* 65, 367–389.

Input:

- `a` : The location parameter.
- `b` : The scale parameter.
- `c` : The shape parameter.
- `d` : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- `x` : The standard deviation of the distribution

See also:

[nb_distribution.skt_mode](#), [nb_distribution.skt_median](#)
[nb_distribution.skt_mean](#), [nb_distribution.skt_variance](#)

Written by Kenneth Åström Paulsen

► **skt_variance ↑**

`x = nb_distribution.skt_variance(a,b,c,d)`

Description:

Variance of the Azzalini skewed t-distribution.

Azzalini, A. and Capitanio,A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew-t distribution. J.Roy.Statist.Soc. B 65, 367-389.

Input:

- a : The location parameter.
- b : The scale parameter.
- c : The shape parameter.
- d : Degrees of freedom (default is positive infinity which corresponds to the skew normal distribution)

Output:

- x : The variance of the distribution

See also:

[nb_distribution.skt_mode](#), [nb_distribution.skt_median](#)
[nb_distribution.skt_mean](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **smoothDensity ↑**

`smoothDensity(obj,smoothing,varargin)`

Description:

Smooth out a distribution of type kernel.

This method should be used with care, as the many of the properties of the original distribution may be changed.

Input:

- obj : A scalar nb_distribution object
- smoothing : > A 1x2 cell : First element is the number of elements to smooth out in the neighbourhood of the selected point, while the second input is the index of selected point.
 - > 'kernel' : Using a kernel smoother on the random draws to smooth out distribution.

Optional input:

```
- varargin : Optional input given to the nb_ksdensity function. Please  
see help for that function.
```

Examples:

```
est = nb_distribution.perc2DistCDF([10,30,50,70,90],[1,4,6,7,8],-1,14);  
s = copy(est);  
smoothDensity(s,'kernel');  
plot([est,s])
```

Written by Kenneth Sæterhagen Paulsen

► **std** ↑

```
x = std(obj)
```

Description:

Evaluate the standard deviation of the given distribution.

The variance is normalized with T-1

Input:

```
- obj : An object of class nb_distribution
```

Output:

```
- x : numel(obj) == 1 : A double with size as 1 x 1  
otherwise : A double with size 1 x nobj
```

Written by Kenneth Sæterhagen Paulsen

► **t_cdf** ↑

```
f = nb_distribution.t_cdf(x,m)  
f = nb_distribution.t_cdf(x,m,a,b)
```

Description:

CDF of the student-t distribution.

Input:

- `x` : The point to evaluate the cdf, as a double
- `m` : The number of degrees of freedom. Must be positive.
- `a` : The location parameter. Optional. Default is 0.
- `b` : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- `f` : The CDF at the evaluated points, same size as `x`.

See also:

[nb_distribution.t_pdf](#), [nb_distribution.t_rand](#)
[nb_distribution.t_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **t_domain** ↑

```
f = nb_distribution.t_domain
```

Description:

Get the domain of the student's t-distribution

Output:

- `domain` : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **t_icdf** ↑

```
x = nb_distribution.t_icdf(p,m)
x = nb_distribution.t_icdf(p,m,a,b)
```

Description:

Returns the inverse of the cdf at `p` of the student-t distribution

Input:

- `p` : A vector of probabilities
- `m` : The number of degrees of freedom. Must be positive.
- `a` : The location parameter. Optional. Default is 0.
- `b` : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : A double with the quantile at each element of p of the student-t distribution

See also:

[nb_distribution.t_cdf](#), [nb_distribution.t_rand](#)
[nb_distribution.t_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► t_kurtosis ↑

```
x = nb_distribution.t_kurtosis(m)
x = nb_distribution.t_kurtosis(m,a,b)
```

Description:

Kurtosis of the student-t distribution. Only defined for $m > 2$, otherwise not defined (will return nan).

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The kurtosis of the student-t distribution

See also:

[nb_distribution.f_median](#), [nb_distribution.f_mean](#)
[nb_distribution.f_variance](#), [nb_distribution.f_skewness](#)

Written by Kenneth Sæterhagen Paulsen

► t_mean ↑

```
x = nb_distribution.t_mean(m)
x = nb_distribution.t_mean(m,a,b)
```

Description:

Mean of the student-t distribution. Only defined for $m > 1$, otherwise not defined (will return nan).

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The mean of the student-t distribution

See also:

[nb_distribution.t_mode](#), [nb_distribution.t_median](#)
[nb_distribution.t_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **t_median** ↑

```
x = nb_distribution.t_median(m)
x = nb_distribution.t_median(m,a,b)
```

Description:

Median of the student-t distribution

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The median of the student-t distribution

See also:

[nb_distribution.t_mode](#), [nb_distribution.t_mean](#)
[nb_distribution.t_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **t_mode** ↑

```
x = nb_distribution.t_mode(m)
x = nb_distribution.t_mode(m, a, b)
```

Description:

Mode of the student-t distribution.

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0. Optional. Default is 1.

Output:

- x : The mode of the student-t distribution

See also:

[nb_distribution.t_median](#), [nb_distribution.t_mean](#)
[nb_distribution.t_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **t_pdf** ↑

```
f = nb_distribution.t_pdf(x, m)
f = nb_distribution.t_pdf(x, m, a, b)
```

Description:

PDF of the student-t distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0. Optional. Default is 1.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.t_cdf](#), [nb_distribution.t_rand](#)
[nb_distribution.t_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **t_rand** ↑

```
draws = nb_distribution.t_rand(nrow, ncol, m)
draws = nb_distribution.t_rand(nrow, ncol, m, a, b)
```

Description:

Draw random numbers from the student-t distribution

Input:

- nrow : Number of rows of the matrix drawn from the student-t distribution
- ncol : Number of columns of the matrix drawn from the student-t distribution
- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0. Optional. Default is 1.

Output:

- draws : A nrow x ncol matrix of random numbers from the student-t distribution

See also:

[nb_distribution.t_pdf](#), [nb_distribution.t_cdf](#)

Written by Kenneth S. Paulsen

► **t_skewness** ↑

```
x = nb_distribution.t_skewness(m)
x = nb_distribution.t_skewness(m, a, b)
```

Description:

Skewness of the student-t distribution. Only defined for $m > 3$. Will return nan otherwise.

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The skewness of the student-t distribution

See also:

[nb_distribution.t_median](#), [nb_distribution.t_mean](#)
[nb_distribution.t_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **t_std ↑**

```
x = nb_distribution.t_std(m)
x = nb_distribution.t_std(m,a,b)
```

Description:

Standard deviation of the student-t distribution. Only defined for $m > 2$. Will return nan otherwise.

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The standard deviation of the student-t distribution

See also:

[nb_distribution.t_mode](#), [nb_distribution.t_median](#)
[nb_distribution.t_mean](#), [nb_distribution.t_variance](#)

► **t_variance ↑**

```
x = nb_distribution.t_variance(m)
x = nb_distribution.t_variance(m,a,b)
```

Description:

Variance of the student-t distribution. Only defined for $m > 1$. Will return nan otherwise.

Input:

- m : The number of degrees of freedom. Must be positive.
- a : The location parameter. Optional. Default is 0.
- b : The scale parameter. Must be > 0 . Optional. Default is 1.

Output:

- x : The variance of the student-t distribution

See also:

[nb_distribution.t_mode](#), [nb_distribution.t_median](#)
[nb_distribution.t_mean](#)

► **tri_cdf ↑**

```
f = nb_distribution.tri_cdf(x,m,k,d)
```

Description:

CDF of the triangular distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.tri_pdf](#), [nb_distribution.tri_rand](#)
[nb_distribution.tri_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_domain** ↑

f = nb_distribution.tri_domain(m,k,d)

Description:

Get the domain of the triangular distribution

Inputs:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **tri_icdf** ↑

x = nb_distribution.tri_icdf(p,m,k,d)

Description:

Returns the inverse of the cdf at p of the triangular distribution

Input:

- p : A vector of probabilities
- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- `x` : A double with the quantile at each element of `p` of the triangular distribution

See also:

[nb_distribution.tri_cdf](#), [nb_distribution.tri_rand](#)
[nb_distribution.tri_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_kurtosis** ↑

`x = nb_distribution.tri_kurtosis(m, k, d)`

Description:

Kurtosis of the triangular distribution

Input:

- `m` : Lower bound of the triangular distribution.
- `k` : Upper bound of the triangular distribution.
- `d` : Mode of the triangular distribution.

Output:

- `x` : The kurtosis of the triangular distribution

See also:

[nb_distribution.tri_median](#), [nb_distribution.tri_mean](#)
[nb_distribution.tri_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_mean** ↑

`x = nb_distribution.tri_mean(m, k, d)`

Description:

Mean of the triangular distribution

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The mean of the triangular distribution

See also:

[nb_distribution.tri_mode](#), [nb_distribution.tri_median](#)
[nb_distribution.tri_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_median** ↑

```
x = nb_distribution.tri_median(m, k, d)
```

Description:

Median of the triangular distribution

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The median of the triangular distribution

See also:

[nb_distribution.tri_mode](#), [nb_distribution.tri_mean](#)
[nb_distribution.tri_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_mode** ↑

```
x = nb_distribution.tri_mode(m, k, d)
```

Description:

Mode of the triangular distribution.

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The mode of the triangular distribution

See also:

[nb_distribution.tri_median](#), [nb_distribution.tri_mean](#)
[nb_distribution.tri_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [tri_pdf ↑](#)

```
f = nb_distribution.tri_pdf(x, m, k, d)
```

Description:

PDF of the triangular distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.tri_cdf](#), [nb_distribution.tri_rand](#)

[nb_distribution.tri_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **tri_rand** ↑

```
draws = nb_distribution.tri_rand(nrow,ncol,m,k,d)
```

Description:

Draw random numbers from the triangular distribution

Input:

- nrow : Number of rows of the matrix drawn from the triangular distribution
- ncol : Number of columns of the matrix drawn from the triangular distribution
- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- draws : A nrow x ncol matrix of random numbers from the triangular distribution

See also:

[nb_distribution.tri_pdf](#), [nb_distribution.tri_cdf](#)

Written by Kenneth S. Paulsen

► **tri_skewness** ↑

```
x = nb_distribution.tri_skewness(m,k,d)
```

Description:

Skewness of the triangular distribution

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The skewness of the triangular distribution

See also:

[nb_distribution.tri_median](#), [nb_distribution.tri_mean](#)
[nb_distribution.tri_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **tri_std ↑**

x = nb_distribution.tri_std(m,k,d)

Description:

Standard deviation of the triangular distribution

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The standard deviation of the triangular distribution

See also:

[nb_distribution.tri_mode](#), [nb_distribution.tri_median](#)
[nb_distribution.tri_mean](#), [nb_distribution.tri_variance](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **tri_variance ↑**

x = nb_distribution.tri_variance(m,k,d)

Description:

Variance of the triangular distribution

Input:

- m : Lower bound of the triangular distribution.
- k : Upper bound of the triangular distribution.
- d : Mode of the triangular distribution.

Output:

- x : The variance of the triangular distribution

See also:

[nb_distribution.tri_mode](#), [nb_distribution.tri_median](#)
[nb_distribution.tri_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_cdf ↑**

f = nb_distribution.truncated_cdf(x,dist,param,lb,ub)

Description:

CDF of the truncated distribution

Input:

- x : The point to evaluate the cdf, as a double
- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- f : The CDF at the evaluated points, same size as x.

See also:

[nb_distribution.truncated_pdf](#), [nb_distribution.truncated_rand](#)

`nb_distribution.truncated_icdf`

Written by Kenneth Sæterhagen Paulsen

► **truncated_domain** ↑

```
f = nb_distribution.truncated_domain(dist,param,lb,ub)
```

Description:

Get the domain of the truncated distribution

Inputs:

- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.
- `ub` : Upper bound of the truncated distribution.

Output:

- `domain` : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **truncated_icdf** ↑

```
x = nb_distribution.truncated_icdf(p,dist,param,lb,ub)
```

Description:

Returns the inverse of the cdf at `p` of the truncated distribution

Input:

- `p` : A vector of probabilities
- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.
- `ub` : Upper bound of the truncated distribution.

Output:

- `x` : A double with the quantile at each element of `p` of the truncated distribution

See also:

[nb_distribution.truncated_cdf](#), [nb_distribution.truncated_rand](#)
[nb_distribution.truncated_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_kurtosis** ↑

`x = nb_distribution.truncated_kurtosis(dist,param,lb,ub)`

Description:

Kurtosis of the truncated distribution. The calculation is simulation based and may vary. Set seed to prevent this, or see the kurtosis method of the `nb_distribution` class.

Input:

- `dist` : The name of the underlying distribution as a string. Must be supported by the `nb_distribution` class.
- `param` : A cell with the parameters of the selected distribution.
- `lb` : Lower bound of the truncated distribution.
- `ub` : Upper bound of the truncated distribution.

Output:

- `x` : The kurtosis of the truncated distribution

See also:

[nb_distribution.truncated_median](#), [nb_distribution.truncated_mean](#)
[nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_mean** ↑

`x = nb_distribution.truncated_mean(dist,param,lb,ub)`

Description:

Mean of the truncated distribution. The calculation is simulation based and may vary, except for truncated normal. Set seed to prevent this, or see the mean method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The mean of the truncated distribution

See also:

[nb_distribution.truncated_mode](#), [nb_distribution.truncated_median](#)
[nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_median ↑**

x = nb_distribution.truncated_median(dist,param,lb,ub)

Description:

Median of the truncated distribution.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The median of the truncated distribution

See also:

[nb_distribution.truncated_mode](#), [nb_distribution.truncated_mean](#)
[nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_mode** ↑

```
x = nb_distribution.truncated_mode(dist,param,lb,ub)
```

Description:

Mode of the truncated distribution.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The mode of the truncated distribution

See also:

[nb_distribution.truncated_median](#), [nb_distribution.truncated_mean](#)
[nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_pdf** ↑

```
f = nb_distribution.truncated_pdf(x,dist,param,lb,ub)
```

Description:

PDF of the truncated distribution

Input:

```
- x      : The point to evaluate the pdf, as a double
- dist   : The name of the underlying distribution as a string. Must be
           supported by the nb_distribution class.
- param  : A cell with the parameters of the selected distribution.
- lb     : Lower bound of the truncated distribution.
- ub     : Upper bound of the truncated distribution.
```

Output:

```
- f : The PDF at the evaluated points, same size as x.
```

See also:

[nb_distribution.truncated_cdf](#), [nb_distribution.truncated_rand](#)
[nb_distribution.truncated_icdf](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **truncated_rand** ↑

```
draws = nb_distribution.truncated_rand(nrow, ncol, dist, param, lb, ub)
```

Description:

Draw random number from a truncated distribution.

Input:

```
- nrow  : Number of rows of the matrix drawn from the student-t
          distribution
- ncol  : Number of columns of the matrix drawn from the student-t
          distribution
- dist   : The name of the underlying distribution as a string. Must be
           supported by the nb_distribution class.
- param  : A cell with the parameters of the selected distribution.
- lb     : Lower bound of the truncated distribution.
- ub     : Upper bound of the truncated distribution.
```

Output:

```
- draws : A nrow x ncol matrix of random numbers from the truncated
          distribution
```

Written by Kenneth SÃ¶terhagen Paulsen

► **truncated_skewness** ↑

```
x = nb_distribution.truncated_skewness(dist,param,lb,ub)
```

Description:

Skewness of the truncated distribution. The calculation is simulation based and may vary. Set seed to prevent this, or see the skewness method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The skewness of the truncated distribution

See also:

[nb_distribution.truncated_median](#), [nb_distribution.truncated_mean](#)
[nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_std** ↑

```
x = nb_distribution.truncated_std(dist,param,lb,ub)
```

Description:

Standard deviation of the truncated distribution

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The standard deviation of the truncated distribution

See also:

[nb_distribution.truncated_mode](#), [nb_distribution.truncated_median](#)
[nb_distribution.truncated_mean](#), [nb_distribution.truncated_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncated_variance ↑**

```
x = nb_distribution.truncated_variance(dist,param,lb,ub)
```

Description:

Variance of the truncated distribution. The calculation is simulation based and may vary, except for truncated normal. Set seed to prevent this, or see the variance method of the nb_distribution class.

Input:

- dist : The name of the underlying distribution as a string. Must be supported by the nb_distribution class.
- param : A cell with the parameters of the selected distribution.
- lb : Lower bound of the truncated distribution.
- ub : Upper bound of the truncated distribution.

Output:

- x : The variance of the truncated distribution

See also:

[nb_distribution.truncated_mode](#), [nb_distribution.truncated_median](#)
[nb_distribution.truncated_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **truncgamma_mean ↑**

```
x = nb_distribution.truncgamma_mean(m,k,lb,ub)
```

Description:

Mean of the truncated gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m*k$
- k : A parameter such that the variance of the gamma = $m*(k^2)$
- lb : Lower bound
- ub : Upper bound

Output:

- x : The mean of the truncated gamma distribution

See also:

[nb_distribution.truncgamma_variance](#)

Written by Kenneth Sæterhagen Paulsen

► [truncgamma_variance ↑](#)

x = nb_distribution.truncgamma_variance(m, k)

Description:

Variance of the truncated gamma distribution

Input:

- m : A parameter such that the mean of the gamma = $m*k$
- k : A parameter such that the variance of the gamma = $m*(k^2)$

Output:

- x : The variance of the truncated gamma distribution

See also:

[nb_distribution.truncgamma_mean](#)

Written by Kenneth Sæterhagen Paulsen

► [truncnormal_mean ↑](#)

```
x = nb_distribution.truncnormal_mean(m, k, lb, ub)
```

Description:

Mean of the truncated normal distribution

Input:

- m : A parameter such that the mean of the normal is m
- k : A parameter such that the std of the normal is k
- lb : Lower bound
- ub : Upper bound

Output:

- x : The mean of the truncated normal distribution

See also:

[nb_distribution.truncnormal_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **truncnormal_variance ↑**

```
x = nb_distribution.truncnormal_variance(m, k, lb, ub)
```

Description:

Variance of the truncated normal distribution

Input:

- m : A parameter such that the mean of the normal = m
- k : A parameter such that the std of the normal = k

Output:

- x : The variance of the truncated normal distribution

See also:

[nb_distribution.truncnormal_mean](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_cdf** ↑

```
f = nb_distribution.uniform_cdf(x,m,k)
```

Description:

CDF of the uniform distribution

Input:

- *x* : The point to evaluate the cdf, as a double
- *m* : Lower limit of the support.
- *k* : Upper limit of the support.

Output:

- *f* : The CDF at the evaluated points, same size as *x*.

See also:

[nb_distribution.uniform_pdf](#), [nb_distribution.uniform_rand](#)
[nb_distribution.uniform_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_domain** ↑

```
f = nb_distribution.uniform_domain
```

Description:

Get the domain of the uniform distribution

Inputs:

- *m* : Lower limit of the support.
- *k* : Upper limit of the support.

Output:

- *domain* : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **uniform_icdf** ↑

```
x = nb_distribution.uniform_icdf(p,m,k)
```

Description:

Returns the inverse of the cdf at p of the uniform(m,k) distribution

Input:

- p : A vector of probabilities
- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : A double with the quantile at each element of p of the uniform(m,k) distribution

See also:

[nb_distribution.uniform_cdf](#), [nb_distribution.uniform_rand](#)
[nb_distribution.uniform_pdf](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_kurtosis** ↑

```
x = nb_distribution.uniform_kurtosis(m,k)
```

Description:

Kurtosis of the uniform distribution

Input:

- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : The kurtosis of the uniform distribution

See also:

[nb_distribution.uniform_median](#), [nb_distribution.uniform_mean](#)
[nb_distribution.uniform_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_mean** ↑

`x = nb_distribution.uniform_mean(m, k)`

Description:

Mean of the uniform distribution

Input:

- `m` : Lower limit of the support.
- `k` : Upper limit of the support.

Output:

- `x` : The mean of the uniform distribution

See also:

[nb_distribution.uniform_mode](#), [nb_distribution.uniform_median](#)
[nb_distribution.uniform_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_median** ↑

`x = nb_distribution.uniform_median(m, k)`

Description:

Median of the uniform distribution

Input:

- `m` : Lower limit of the support.
- `k` : Upper limit of the support.

Output:

- `x` : The median of the uniform distribution

See also:

[nb_distribution.uniform_mode](#), [nb_distribution.uniform_mean](#)
[nb_distribution.uniform_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_mode** ↑

`x = nb_distribution.uniform_mode(m, k)`

Description:

Mode of the uniform distribution. Any point between m and k . Return the mid point (mean).

Input:

- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : The mode of the uniform distribution

See also:

[nb_distribution.uniform_median](#), [nb_distribution.uniform_mean](#)
[nb_distribution.uniform_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **uniform_pdf** ↑

`f = nb_distribution.uniform_pdf(x, m, k)`

Description:

PDF of the uniform distribution

Input:

- x : The point to evaluate the pdf, as a double
- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.uniform_cdf](#), [nb_distribution.uniform_rand](#)
[nb_distribution.uniform_icdf](#)

Written by Kenneth Sætherhagen Paulsen

► **uniform_rand** ↑

```
draws = nb_distribution.uniform_rand(nrow, ncol, m, k)
```

Description:

Draw random numbers from the uniform distribution

Input:

- nrow : Number of rows of the matrix drawn from the uniform distribution.
- ncol : Number of columns of the matrix drawn from the uniform distribution.
- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- draws : A nrow x ncol matrix of random numbers from the uniform distribution

See also:

[nb_distribution.uniform_pdf](#), [nb_distribution.uniform_cdf](#)

Written by Kenneth S. Paulsen

► **uniform_skewness** ↑

```
x = nb_distribution.uniform_skewness(m, k)
```

Description:

Skewness of the uniform distribution

Input:

- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : The skewness of the uniform distribution

See also:

[nb_distribution.uniform_median](#), [nb_distribution.uniform_mean](#)
[nb_distribution.uniform_variance](#)

Written by Kenneth SÃ¶therhagen Paulsen

► **uniform_std** ↑

x = nb_distribution.uniform_std(m, k)

Description:

Standard deviation of the uniform distribution

Input:

- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : The standard deviation of the uniform distribution

See also:

[nb_distribution.uniform_mode](#), [nb_distribution.uniform_median](#)
[nb_distribution.uniform_mean](#), [nb_distribution.uniform_variance](#)

Written by Kenneth SÃ¶therhagen Paulsen

► **uniform_variance** ↑

x = nb_distribution.uniform_variance(m, k)

Description:

Variance of the uniform distribution

Input:

- m : Lower limit of the support.
- k : Upper limit of the support.

Output:

- x : The variance of the uniform distribution

See also:

[nb_distribution.uniform_mode](#), [nb_distribution.uniform_median](#)
[nb_distribution.uniform_mean](#)

Written by Kenneth Sæterhagen Paulsen

► var ↑

x = var(obj)

Description:

Re-directs to the variance method.

Written by Kenneth Sæterhagen Paulsen

► variance ↑

x = variance(obj)

Description:

Evaluate the variance of the given distribution.

The variance is normalized with T-1

Input:

- obj : An object of class nb_distribution

Output:

```
- x      : numel(obj) == 1 : A double with size as 1 x 1
           otherwise       : A double with size 1 x nobj
```

Written by Kenneth SÃ¶terhagen Paulsen

► **vertcat** ↑

```
obj = vertcat(varargin)
```

Description:

Vertical concatenation

Input:

```
- varargin : Either an object of class nb_distribution or an object of
            class double
```

Output:

```
- obj      : An object of class nb_distribution
```

Written by Kenneth SÃ¶terhagen Paulsen

► **wald_cdf** ↑

```
f = nb_distribution.wald_cdf(x,m,k)
```

Description:

CDF of the wald distribution

Input:

```
- x : The point to evaluate the cdf, as a double
- m : The first parameter of the wald distribution. Must be a 1x1 double
      greater than 0. E[X] = m.
- k : The second parameter of the wald distribution. Must be a 1x1 double
      greater than 0.
```

Output:

```
- f : The CDF at the evaluated points, same size as x.
```

See also:

[nb_distribution.wald_pdf](#), [nb_distribution.wald_rand](#)
[nb_distribution.wald_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_domain** ↑

f = nb_distribution.wald_domain

Description:

Get the domain of the wald distribution

Output:

- domain : A 1x2 double with the lower and upper limits of the domain.

Written by Kenneth Sæterhagen Paulsen

► **wald_icdf** ↑

x = nb_distribution.wald_icdf(p,m,k)

Description:

Inverse CDF of the wald density

Input:

- p : A vector of probabilities

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.

- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The inverse CDF at the evaluated probabilities, same size as p.

See also:

[nb_distribution.wald_pdf](#), [nb_distribution.wald_rand](#)
[nb_distribution.wald_cdf](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_kurtosis** ↑

```
x = nb_distribution.wald_kurtosis(m,k)
```

Description:

Kurtosis of the wald distribution

Input:

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The kurtosis of the wald distribution

See also:

[nb_distribution.wald_median](#), [nb_distribution.wald_mean](#)
[nb_distribution.wald_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_mean** ↑

```
x = nb_distribution.wald_mean(m,k)
```

Description:

Mean of the wald distribution

Input:

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The mean of the wald distribution

See also:

`nb_distribution.wald_mode`, `nb_distribution.wald_median`
`nb_distribution.wald_variance`

Written by Kenneth Sæterhagen Paulsen

► **wald_median** ↑

`x = nb_distribution.wald_median(m, k)`

Description:

Median of the wald distribution

Input:

- `m` : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- `k` : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- `x` : The median of the wald distribution

See also:

`nb_distribution.wald_mode`, `nb_distribution.wald_mean`
`nb_distribution.wald_variance`

Written by Kenneth Sæterhagen Paulsen

► **wald_mode** ↑

`x = nb_distribution.wald_mode(m, k)`

Description:

Mode of the wald distribution

Input:

- `m` : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- `k` : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The mode of the wald distribution

See also:

[nb_distribution.wald_median](#), [nb_distribution.wald_mean](#)
[nb_distribution.wald_variance](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_pdf** ↑

f = nb_distribution.wald_pdf(x,m,k)

Description:

PDF of the wald distribution

Input:

- x : The point to evaluate the cdf, as a double
- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- f : The PDF at the evaluated points, same size as x.

See also:

[nb_distribution.wald_cdf](#), [nb_distribution.wald_rand](#)
[nb_distribution.wald_icdf](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_rand** ↑

draws = nb_distribution.wald_rand(nrow,ncol,m,k)

Description:

Draw random numbers from the wald distribution

Input:

- nrow : Number of rows of the matrix drawn from the wald distribution
- ncol : Number of columns of the matrix drawn from the wald distribution
- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- draws : A nrow x ncol matrix of random numbers from the wald distribution

See also:

[nb_distribution.wald_pdf](#), [nb_distribution.wald_cdf](#)

Written by Kenneth S. Paulsen

► **wald_skewness ↑**

`x = nb_distribution.wald_skewness(m, k)`

Description:

Skewness of the wald distribution

Input:

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The skewness of the wald distribution

See also:

[nb_distribution.wald_median](#), [nb_distribution.wald_mean](#)
[nb_distribution.wald_variance](#)

Written by Kenneth Sætherhagen Paulsen

► **wald_std** ↑

```
x = nb_distribution.wald_std(m,k)
```

Description:

Standard deviation of the wald distribution

Input:

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The standard deviation of the wald distribution

See also:

[nb_distribution.wald_mode](#), [nb_distribution.wald_median](#)
[nb_distribution.wald_mean](#), [nb_distribution.wald_std](#)

Written by Kenneth Sæterhagen Paulsen

► **wald_variance** ↑

```
x = nb_distribution.wald_variance(m,k)
```

Description:

Variance of the wald distribution

Input:

- m : The first parameter of the wald distribution. Must be a 1x1 double greater than 0. $E[X] = m$.
- k : The second parameter of the wald distribution. Must be a 1x1 double greater than 0.

Output:

- x : The variance of the wald distribution

See also:

```
nb_distribution.wald_mode, nb_distribution.wald_median  
nb_distribution.wald_mean
```

Written by Kenneth SÃ¶terhagen Paulsen

► **wish_rand** ↑

```
A = wish_rand(h,n)
```

Description:

Draws an $m \times m$ matrix from a wishart distribution with scale matrix h and degrees of freedom $\nu = n$. This procedure uses Bartlett's decomposition.

Note: Parameterized so that mean is $n \cdot h$

Input:

- h : $m \times m$ scale matrix.
- n : scalar degrees of freedom.

Output:

- A : $m \times m$ matrix draw from the wishart distribution.

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_ast_b**

```
f = nb_ast_b(x)
```

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_ast_cstar**

```
cstar = nb_ast_cstar(c,d,e)
```

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_ast_k**

```
f = nb_ast_k(x)
```

Written by Kenneth SÃ¶terhagen Paulsen

◆ nb_chol

```
C = nb_chol(A,type)
```

Description:

Cholesky decomposition. If type is given as 'cov' it will do a eigenvalue decomposition such that $C \cdot C' = A$. In the last case the matrix A must be positive semi-definite.

Input:

- A : A symmetric positive (semi-)definite matrix
- type : Give 'cov' to use eigenvalue decomposition if cholesky decomposition fails. C is not necessary symmetric in this case!
Give 'covrepair' to reset all eigenvalue that are negative to 0.

Output:

- C : A matrix such that $C \cdot C' = A$

Written by Kenneth Sæterhagen Paulsen

◆ nb_copulacondrnd

```
U = nb_copulacondrnd(M,P,sigma,indC,a,type)
```

Description:

Generate random variables from the conditional gaussian copula.

Input:

- M : The number of simulated observations of each variables.
- P : The number of simulated observations of each variables. (Added as a page)
- sigma : The correlation matrix. As a $N \times N$ double, with ones along the diagonal.
- indC : Index for the variables to condition on. As a $1 \times N$ or $N \times 1$ logical. sum(indC) must equal Q.
- a : The values to condition on. As $Q \times 1$ double.
- type : Set the transformation method to use to transform the rank correlation coefficients to the linear correlation coefficients.

- 'none' : If the correlation in the data is calculated based on the linear correlation coefficients. Default.
- 'spearman' : Uses the formula $\rho = 2 \cdot \sin(\sigma \cdot \pi / 6)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X,'type','spearman')` function.
- 'kendall' : Uses the formula $\rho = \sin(\sigma \cdot \pi / 2)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X,'type','kendall')` function.

The rank correlation coefficients are approximately invariant to the choice of marginal distribution used to transform the draws from the copula to the draws of the different marginals.

Output:

- U : A M x N-Q x P matrix.

Examples:

```
U = nb_copularnd([1,0.5;0.5,1],'spearman');
```

See also:

[nb_copula](#)

Written by Kenneth Sølnerhagen Paulsen

◆ [nb_copularnd](#)

```
U = nb_copularnd(M,P,sigma,type)
```

Description:

Generate random variables from the gaussian copula.

Input:

- M : The number of simulated observations of each variables.
- P : The number of simulated observations of each variables. (Added as a page)
- sigma : The correlation matrix. As a N x N double, with ones along the diagonal.
- type : Set the transformation method to use to transform the rank correlation coefficients to the linear correlation coefficients.

- 'none' : If the correlation in the data is calculated based on the linear correlation coefficients. Default.
- 'spearman' : Uses the formula $\rho = 2 \cdot \sin(\sigma \cdot \pi / 6)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X,'type','spearman')` function.
- 'kendall' : Uses the formula $\rho = \sin(\sigma \cdot \pi / 2)$. Can be used if the rank correlation coefficients are calculated based on the data series. Using the `corr(X,'type','kendall')` function.

The rank correlation coefficients are approximately invariant to the choice of marginal distribution used to transform the draws from the copula to the draws of the different marginals.

Output:

- U : A M x N x P matrix.

Examples:

```
U = nb_copularnd([1,0.5;0.5,1],'spearman');
```

See also:

[nb_copula](#)

Written by Kenneth Åkerhagen Paulsen

◆ nb_covShrinkDiag

```
[C,lambda]= nb_covShrinkDiag(x,lambda)
```

Description:

Let X be a data matrix with size T x N. This function estimate the covariance matrix of the data X using a shrinkage against the diagonal matrix of the variances of the data in X.

```
C = lambda*D + (1-lambda)*S
```

where D is the diagonal matrix and S is the normal estimate of the covariance matrix $((X'*X)/(T-1))$

References:

Kwan (2011), "An Introduction to Shrinkage Estimation of the Covariance Matrix: A Pedagogic Illustration"

Input:

- X : A T x N double where T is the number of observations and N is the number of variables.
- lambda : Give this input, if you want to set the shrinkage parameter based on judgment. Must in this case be a number between 0 and 1. Default is to estimate lambda as in Kwan (2011). 0 means no shrinkage.

Output:

- C : The estimate covariance matrix.
- lambda : The calculated shrinkage parameter.

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

◆ nb_covrepair

C = nb_covrepair(A,type)

Description:

Convert a non (semi-)definite matrix to (semi-)definite matrix.

Input:

- A : A symmetric matrix.
- type : true or false. Give true to find the closest semi-definite matrix. Default is false, i.e. to find the closest definite matrix.

Output:

- C : A symmetric matrix with same size as A.

Written by Kenneth Sæterhagen Paulsen

◆ nb_drawCov

C = nb_drawCov(N,P)

Description:

Draw random correlation matrices of size N x N.

Input:

- N : The size of the correlation matrix.
- draws : Number of draws. Added as separate pages.

Output:

- C : A N x N x draws double.

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_fStatPValue**

p = nb_fStatPValue(x, m, k)

Description:

P-value of F-statistics.

Input:

- x : The point to evaluate the t-statistics at, as a double.
- m : Numerator degree of freedom. Must be positive.
- k : Denominator degree of freedom. Must be positive.

Output:

- p : P-value of F-statistics, same size as x.

See also:

[nb_distribution.f_pdf](#), [nb_distribution.f_rand](#)
[nb_distribution.f_icdf](#), [nb_distribution.f_cdf](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_invppsi**

y = nb_invppsi(x, err)

Description:

Inverse of psi function using a newton algorithm.

Input:

```
- x : A double.  
- err : The stopping criterion for the newton algorithm.
```

Output:

```
- y : A double.
```

See also:

[psi](#)

Kenneth Sæterhagen Paulsen

◆ nb_ksdensity

```
[f,x,u] = nb_ksdensity(yData,varargin)  
[f,x,u] = nb_ksdensity(yData,domain,varargin)
```

Description:

Kernel density estimation.

Input:

```
- yData : A numVar x numSim double.  
- domain : Same as the 'domain' input.
```

Optional input:

```
- 'domain' : Either a 1 x numDomain or a numVar x numDomain double  
with the point for where to evaluate the function f.  
Can be empty, and in this case it will be estimated.  
  
- 'bandWidth' : Sets the bandwidth to use during kernel density  
estimation. If given as a number less than or equal to  
0, the default value is used; sigma*(4/(3*N))^(1/5),  
where sigma is the standard deviation of the xi input  
calculated using mean absolute deviation and N is the  
size(yData,2). The default value is found for each  
function separately (numVar).  
  
- 'numDomain' : This option will set the number of values for where the  
function f are to be evaluated if domain is not provided.  
Default is 1000.
```

Output:

```
- f : A numVar x numDomain double with the estimated density function.  
- x : A numVar x numDomain double with the domain of the density  
function f.  
- u : A numVar x 1 double with the selected band widths.
```

See also:

[ksdensity](#), [Distribution.ksdensity](#), [Distribution.ksdomain](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_lognormal_deriv**

```
x = nb_lognormal_deriv(x,m,k)
```

Description:

Derivate of the log normal distribution.

Input:

- x : The point to evaluate the derivative of the log normal PDF, as a double.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$
- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$

Output:

- x : The derivatives of the log normal PDF at the evaluated points, same size as x.

See also:

[nb_distribution](#), [nb_distribution.lognormal_pdf](#)
[nb_distribution.lognormal_cdf](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mci**

```
x = nb_mci(func,a,b,draws,dist)
```

Description:

Integrate a univariate function from a to b using a monte carlo method.

Input:

- func : A function handle.
- a : Lower limit, as a scalar double. Can be -inf.
- b : Upper limit, as a scalar double. Can be inf.

- draws : Number of draws. As a number grater than or equal to 1000. Default is 10000.
- dist : The distribution to use to do the montecarlo integration. As a nb_distribution object. Default is to use the uniform(a,b) distribution. This distribution will be truncated in the interval [a,b], so this means that the domain of the given distribution must contain the interval [a,b].

Caution : If a == -inf and/or b == inf. The truncated normal distribution is used as the default distribution.

Output:

- x : The value of the integral.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mode**

```
m = nb_mode(x,dim,method,varargin)
```

Description:

Find the mode of the empirical distribution of the number in x along the wanted dimension.

Input:

- x : A 3D double matrix.
- dim : Either 1, 2 or 3. 1 is default.
- method : A string:
 - > 'kernel' : Using kernel density estimation and finds the mode of the estimated distribution. (Default)

To decrease the std in the mode estimate the 'width' input can be used. The default is

```
sig = median(abs(x-median(x))) / 0.6745;
w    = 2 * sig * (4/(3*N))^(1/5);
```
- varargin : Extra inputs to the differen mode estimators
 - > 'kernel' :

Optional input given to the nb_ksdensity function. Please see help for that function.
 - > The rest of the methods have no optional inputs.

Output:

- m : The mode with size matching the inputs x and dim.

Examples:

```
m = nb_mode(randn(1000,1),1,'kernel')
```

See also:

[mode](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_mvExpNorm**

```
x = nb_mvExpNorm(n,sigma)
```

Description:

Calculates the expectation of the 2-norm of a multivariate normal distribution with diagonal covariance matrix and where all the diagonal elements has the same value; sigma.

It is also assumed that the distribution has mean zero for all elements.

Input:

- n : The number of variables. As a positive integer.
- sigma : The standard deviation of all the variables. As a positive double.

Output:

- x : The expected 2-norm of the distribution. As a scalar double.

Written by Kenneth Sætherhagen Paulsen

◆ **nb_mvncdf**

```
p = nb_mvncdf(x,mu,sigma)
```

Description:

Evaluate multivariate normal cdf

This code uses the qsimvn made by;

Alan Genz, WSU Math, PO Box 643113, Pullman, WA 99164-3113

Input:

```
- x      : A nPoints x nVar double with the points to evaluate.  
- mu    : A 1 x nVar double with the mean of the different variables of the  
          distribution.  
- sigma : A nVar x nVar double with the correlation matrix.
```

Output:

p : A nPoints x 1 double with the evaluated probabilities.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mvnrnd**

U = nb_mvnrnd(M,P,mu,sigma,tol)

Description:

Draw random variables from the multivariate normal distribution.

Input:

```
- M      : The number of simulate observation of each variables.  
- P      : The number of simulate observation of each variables. (Added  
          as a page)  
- mu    : The mean of the variables. As a 1 x N double.  
- sigma : The covariance matrix. As a N x N double. Must be symmetric.
```

Caution: Notice that this function compared to mvnrnd made by MATLAB tries to repair the covariance matrix if it is not positive semi-definite. This it does by setting all eigenvalues of the sigma matrix that are negative to zero.

- tol : The tolerance when checking for symmetry of sigma.

Output:

- U : A M x N x P matrix.

See also:

[nb_chol](#), [nb_mvtnrand](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mvnrndChol**

```
U = nb_mvnrndChol(M,P,mu,T)
```

Description:

Draw random variables from the multivariate normal distribution.

Input:

- M : The number of simulate observation of each variables.
- P : The number of simulate observation of each variables. (Added as a page)
- mu : The mean of the variables. As a 1 x N double.
- T : The cholesky factorized correlation matrix. As a N x N double.

Output:

- U : A M x N x P matrix.

See also:

[nb_mvnrnd](#), [nb_chol](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_mvtncondrand**

```
U = nb_mvtncondrand(M,P,mu,sigma,l,u,indC,a,tol)
```

Description:

Draw random variables from the conditional multivariate truncated normal distribution.

Input:

- M : The number of simulate observation of each variables.
- P : The number of simulate observation of each variables. (Added as a page)
- mu : The mean of the variables. As a N x 1 double.
- sigma : The covariance matrix. As a N x N double. Must be symmetric.
- l : Lower bound of the truncated distribution. As a 1 x N double.
May include -inf.
- u : Upper bound of the truncated distribution. As a 1 x N double.
May include inf.

- `indC` : Index for the variables to condition on. As a $1 \times N$ or $N \times 1$ logical. `sum(indC)` must equal Q .
- `a` : The values to condition on. As $Q \times 1$ double.
- `tol` : The tolerance when checking for symmetry of σ .

Output:

- `U` : A $M \times N \times P$ matrix.

See also:

[nb_chol](#), [nb_mvtnrand](#), [nb_mvnrand](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_mvtnrand**

`U = nb_mvtnrand(M,P,mu,sigma,l,u,tol)`

Description:

Draw random variables from the multivariate truncated normal distribution.

This function is based on the `trandn` function written by Zdravko Botev. Available here "https://se.mathworks.com/matlabcentral/fileexchange/53180-truncated-normal-generator?s_tid=prof_contriblnk". Main modifications:

1. Generalized to the non-standardized truncated normal distribution using the Geweke, Hajivassiliou and Keane algorithm.
https://en.wikipedia.org/wiki/GHK_algorithm
2. Allow for simulations of multiple draws from the variables of the distribution.

Reference:

Botev, Z. I. (2016). "The normal law under linear restrictions: simulation and estimation via minimax tilting". Journal of the Royal Statistical Society: Series B (Statistical Methodology). doi:10.1111/rssb.12162

Input:

- `M` : The number of simulate observation of each variables.
- `P` : The number of simulate observation of each variables. (Added as a page)
- `mu` : The mean of the variables. As a $N \times 1$ double.
- `sigma` : The covariance matrix. As a $N \times N$ double. Must be symmetric.
- `l` : Lower bound of the truncated distribution. As a $N \times 1$ double.

May include -inf.

- u : Upper bound of the truncated distribution. As a N x 1 double.
May include inf.
- tol : The tolerance when checking for symmetry of sigma.

Output:

- U : A M x N x P matrix.

See also:

[nb_chol](#), [nb_mvnrnd](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_normal_deriv**

`x = nb_normal_deriv(x,m,k)`

Description:

Derivative of the normal distribution.

Input:

- x : The point to evaluate the derivative of the normal PDF, as a double.
- m : The mean of the distribution.
- k : The std of the distribution.

Output:

- x : The derivatives of the normal PDF at the evaluated points, same size as x.

See also:

[nb_distribution](#), [nb_distribution.normal_pdf](#)
[nb_distribution.normal_cdf](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_tAbsMoment**

`f = nb_tAbsMoment(x,r)`

Description:

Absolute moment of the Student-t distribution.

See Zhu and Galbraith (2009).

Input:

- x : The number of degrees of freedom.
- r : The absolute moment order to calculate.

Output:

- f : The rth order absolute moment

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_tStatPValue**

p = nb_tStatPValue(x, m)

Description:

P-value of t-statistics.

Input:

- x : The point to evaluate the t-statistics at, as a double.
- m : The number of degrees of freedom. Must be positive.

Output:

- p : P-value of t-statistics, same size as x.

See also:

[nb_distribution.t_pdf](#), [nb_distribution.t_rand](#)
[nb_distribution.t_icdf](#), [nb_distribution.t_cdf](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **qsimvn**

[p,e] = qsimvn(m,r,a,b)

Description:

This function uses a randomized quasi-random rule with m points to estimate an MVN probability for positive definite covariance matrix r , with lower integration limits a and upper integration limits b . Probability p is output with error estimate e .

This function uses an algorithm given in the paper
"Numerical Computation of Multivariate Normal Probabilities", in
J. of Computational and Graphical Stat., 1(1992), pp. 141-149, by
Alan Genz, WSU Math, PO Box 643113, Pullman, WA 99164-3113
Email : AlanGenz@wsu.edu

The primary references for the numerical integration are
"On a Number-Theoretical Integration Method"
H. Niederreiter, Aequationes Mathematicae, 8(1972), pp. 304-11, and
"Randomization of Number Theoretic Methods for Multiple Integration"
R. Cranley and T.N.L. Patterson, SIAM J Numer Anal, 13(1976), pp.
904-14.

Examples:

```
r      = [4 3 2 1;3 5 -1 1;2 -1 4 2;1 1 2 5];
a      = -inf*[1 1 1 1]'; b = [ 1 2 3 4 ]';
[p, e] = qsimvn( 5000, r, a, b ); disp([ p e ])
```

Copyright (C) 2013, Alan Genz, All rights reserved.

25.16 Estimation

- nb_addStars
- nb_almonPoly
- nb_arimaEstimator.drawParameters
- nb_arimaEstimator.getCoeff
- nb_arimaEstimator.getMeasurementEqRes
- nb_arimaEstimator.getResidual
- nb_arimaEstimator.makeArtificial
- nb_arimaEstimator.template
- nb_arimaStateSpace
- nb_bVarEstimator.applyDummyPrior
- nb_bVarEstimator.calculateMarginalLikelihood
- nb_bVarEstimator.closeWaitbar
- nb_bVarEstimator.defaultHyperLearningSettings
- nb_bVarEstimator.doBayesianMF
- nb_bVarEstimator.doEmpiricalBayesianMF
- nb_bVarEstimator.dummyNormalizationConstant
- nb_bVarEstimator.estimateARLowFreq
- nb_bVarEstimator.getAllMFVariables
- nb_bVarEstimator.getCoeff
- nb_bVarEstimator.getDummyPriorOptions
- nb_bVarEstimator.getMixing
- nb_bVarEstimator.getResidual
- nb_bVarEstimator.getStateSpace
- nb_bVarEstimator.glp
- nb_bVarEstimator.help
- nb_bVarEstimator.horseshoeGibbs
- nb_bVarEstimator.interpretRScale
- nb_almonLag
- nb_arimaEstimator.bootstrapModel
- nb_arimaEstimator.estimate
- nb_arimaEstimator.getDependent
- nb_arimaEstimator.getPredicted
- nb_arimaEstimator.help
- nb_arimaEstimator.print
- nb_arimaFunc
- nb_bVarEstimator.CCCMSampler
- nb_bVarEstimator.applyMeasurementEqRestriction
- nb_bVarEstimator.calculateRMSE
- nb_bVarEstimator.constructInvWeights
- nb_bVarEstimator.doBayesian
- nb_bVarEstimator.doEmpiricalBayesian
- nb_bVarEstimator.dsge
- nb_bVarEstimator.estimate
- nb_bVarEstimator.expandZR
- nb_bVarEstimator.getAllMFVariablesRec
- nb_bVarEstimator.getDependent
- nb_bVarEstimator.getInitAndHyperParam
- nb_bVarEstimator.getPredicted
- nb_bVarEstimator.getSVDPrior
- nb_bVarEstimator.getStateSpaceUnstable
- nb_bVarEstimator.glpMF
- nb_bVarEstimator.horseshoe
- nb_bVarEstimator.horseshoePrior
- nb_bVarEstimator.inwishart

- nb_bVarEstimator.inwishartGibbs
- nb_bVarEstimator.inwishartMFGibbs
- nb_bVarEstimator.jeffreyMCI
- nb_bVarEstimator.laplaceDensity
- nb_bVarEstimator.minnesota
- nb_bVarEstimator.minnesotaMCI
- nb_bVarEstimator.minnesotaMFGibbs
- nb_bVarEstimator.minnesotaVariancePrior
- nb_bVarEstimator.nwishart
- nb_bVarEstimator.nwishartMF
- nb_bVarEstimator.openWaitbar
- nb_bVarEstimator.priorTemplate
- nb_bVarEstimator.removeZR
- nb_bVarEstimator.setUpPriorLR
- nb_bVarEstimator.template
- nb_bVarEstimator.testLRPriorPhi
- nb_betaLag
- nb_betaProfiling
- nb_dfmemlEstimator.bootstrapModel
- nb_dfmemlEstimator.emlIteration
- nb_dfmemlEstimator.getBQ
- nb_dfmemlEstimator.getDependent
- nb_dfmemlEstimator.getFactors
- nb_dfmemlEstimator.getMapping
- nb_dfmemlEstimator.getObservables
- nb_dfmemlEstimator.getResidual
- nb_dfmemlEstimator.getStateNames
- nb_bVarEstimator.inwishartMF
- nb_bVarEstimator.jeffrey
- nb_bVarEstimator.laplace
- nb_bVarEstimator.logMarginalLikelihood
- nb_bVarEstimator.minnesotaGibbs
- nb_bVarEstimator.minnesotaMF
- nb_bVarEstimator.minnesotaMFGibbs2
- nb_bVarEstimator.notifyWaitbar
- nb_bVarEstimator.nwishartMCI
- nb_bVarEstimator.nwishartMFGibbs
- nb_bVarEstimator.print
- nb_bVarEstimator.recursiveEstimation
- nb_bVarEstimator.setUpPriorDIO
- nb_bVarEstimator.setUpPriorSC
- nb_bVarEstimator.testLRPriorH
- nb_bayesEst
- nb_betaPoly
- nb_checkSample
- nb_dfmemlEstimator.checkConvergence
- nb_dfmemlEstimator.estimate
- nb_dfmemlEstimator.getCoeff
- nb_dfmemlEstimator.getEpsInd
- nb_dfmemlEstimator.getIdiosyncraticMapping
- nb_dfmemlEstimator.getNumberofCoeff
- nb_dfmemlEstimator.getPredicted
- nb_dfmemlEstimator.getSolution
- nb_dfmemlEstimator.help

- nb_dfmemlEstimator.initialize
- nb_dfmemlEstimator.normalEstimation
- nb_dfmemlEstimator.recursiveEstimation
- nb_dfmemlEstimator.template
- nb_dynareEstimator.print
- nb_ecmEstimator.bootstrapModel
- nb_ecmEstimator.getCoeff
- nb_ecmEstimator.getPredicted
- nb_ecmEstimator.help
- nb_ecmEstimator.print
- nb_estimator.addExoLags
- nb_estimator.applyRestrictions
- nb_estimator.checkDOFRecursive
- nb_estimator.correctForMissing
- nb_estimator.correctResultsGivenUnbalanced
- nb_estimator.getBlockExogenousRestrictions
- nb_estimator.getSpecificPriors
- nb_estimator.interpretPrecision
- nb_estimator.openWaitbar
- nb_estimator.printCov
- nb_estimator.set2nan
- nb_estimator.testSample
- nb_exprEstimator.bootstrapModel
- nb_exprEstimator.estimate
- nb_exprEstimator.getDependent
- nb_exprEstimator.getResidual
- nb_exprEstimator.makeArtificial
- nb_dfmemlEstimator.intiKF
- nb_dfmemlEstimator.print
- nb_dfmemlEstimator.removeLeadingAndTrailingNaN
- nb_drawFromPosterior
- nb_ecmEstimator.addECMlags
- nb_ecmEstimator.estimate
- nb_ecmEstimator.getDependent
- nb_ecmEstimator.getResidual
- nb_ecmEstimator.makeArtificial
- nb_ecmEstimator.template
- nb_estimator.addRestrictions
- nb_estimator.checkDOF
- nb_estimator.closeWaitbar
- nb_estimator.correctOptionsForUnbalanced
- nb_estimator.fillInForMissing
- nb_estimator.getBounds
- nb_estimator.getVariables
- nb_estimator.notifyWaitbar
- nb_estimator.print
- nb_estimator.removeLeadingAndTrailingNaN
- nb_estimator.template
- nb_estvarols
- nb_exprEstimator.eqs2funcSub
- nb_exprEstimator.getCoeff
- nb_exprEstimator.getPredicted
- nb_exprEstimator.help
- nb_exprEstimator.modelSelectionAlgorithm

- nb_exprEstimator.print
- nb_exprEstimator.template
- nb_fmEstimator.bootstrapModel
- nb_fmEstimator.estimateFactors
- nb_fmEstimator.getCoeff
- nb_fmEstimator.getFactors
- nb_fmEstimator.getPredicted
- nb_fmEstimator.help
- nb_fmEstimator.normalEstimation
- nb_fmEstimator.normalEstimationFAVAR
- nb_fmEstimator.normalEstimationStepAhead
- nb_fmEstimator.recursiveEstimation
- nb_fmEstimator.recursiveEstimationFAVAR
- nb_fmEstimator.recursiveEstimationStepAhead
- nb_getMidasIndex
- nb_hpEstimator.estimate
- nb_hpEstimator.getCoeff
- nb_hpEstimator.getHP
- nb_hpEstimator.print
- nb_harima
- nb_jeffreyPrior
- nb_kalmanLikelihoodDSGE
- nb_kalmanLikelihoodDiffuseDSGE
- nb_kalmanLikelihoodDiffuseTVPDSGE
- nb_kalmanLikelihoodTVPDSGE
- nb_kalmanLikelihoodUnivariateDSGE
- nb_kalmanLikelihoodUnivariateTVPDSGE
- nb_exprEstimator.secureAllLags
- nb_exprEstimator.varModifications
- nb_fmEstimator.estimate
- nb_fmEstimator.getAllDynamicRegressors
- nb_fmEstimator.getDependent
- nb_fmEstimator.getObservables
- nb_fmEstimator.getResidual
- nb_fmEstimator.modelSelectionAlgorithm
- nb_fmEstimator.normalEstimationDynamic
- nb_fmEstimator.normalEstimationSingleEq
- nb_fmEstimator.print
- nb_fmEstimator.recursiveEstimationDynamic
- nb_fmEstimator.recursiveEstimationSingleEq
- nb_fmEstimator.template
- nb_homoOnlySTD
- nb_hpEstimator.estimateHP
- nb_hpEstimator.getDependent
- nb_hpEstimator.help
- nb_hpEstimator.template
- nb_jeffrey
- nb_kalmanLikelihoodBreakPointDSGE
- nb_kalmanLikelihoodDiffuseBreakPointDSGE
- nb_kalmanLikelihoodDiffuseStochasticTrendDSGE
- nb_kalmanLikelihoodMissingDSGE
- nb_kalmanLikelihoodUnivariateBreakPointDSGE
- nb_kalmanLikelihoodUnivariateStochasticTrendDSGE
- nb_kalmanSmootherAndLikelihood

- nb_kalmanSmootherAndLikelihoodUnivariate
- nb_kalmanSmootherDSGE
- nb_kalmanSmootherDiffuseDSGE
- nb_kalmanSmootherDiffuseTVPDSGE
- nb_kalmanSmootherTVPDSGE
- nb_kalmanSmootherUnivariateDSGE
- nb_kalmanSmootherUnivariateTVPDSGE
- nb_kalmanlikelihood
- nb_kalmansmooth
- nb_lasso
- nb_lasso.display
- nb_lasso.maxIter
- nb_lasso.optimset
- nb_lassoEstimator.bootstrapModel
- nb_lassoEstimator.getCoeff
- nb_lassoEstimator.getPredicted
- nb_lassoEstimator.help
- nb_lassoEstimator.print
- nb_legendrePoly
- nb_midasBootstrap
- nb_midasEstimator.bootstrapModel
- nb_midasEstimator.estimate
- nb_midasEstimator.getDependent
- nb_midasEstimator.getResidual
- nb_midasEstimator.modelSelectionAlgorithm
- nb_midasEstimator.print
- nb_midasEstimator.template
- nb_kalmanSmootherBreakPointDSGE
- nb_kalmanSmootherDiffuseBreakPointDSGE
- nb_kalmanSmootherDiffuseStochasticTrendDSGE
- nb_kalmanSmootherMissingDSGE
- nb_kalmanSmootherUnivariateBreakPointDSGE
- nb_kalmanSmootherUnivariateStochasticTrendDSGE
- nb_kalmanfilter
- nb_kalmanlikelihood_missing
- nb_kalmansmooth_missing
- nb_lasso.beta0
- nb_lasso.displayer
- nb_lasso.mode
- nb_lasso.threshold
- nb_lassoEstimator.estimate
- nb_lassoEstimator.getDependent
- nb_lassoEstimator.getResidual
- nb_lassoEstimator.makeArtificial
- nb_lassoEstimator.template
- nb_loadDraws
- nb_midasEstimator.addLags
- nb_midasEstimator.checkDOFRecursive
- nb_midasEstimator.getCoeff
- nb_midasEstimator.getPredicted
- nb_midasEstimator.help
- nb_midasEstimator.normalEstimation
- nb_midasEstimator.recursiveEstimation
- nb_midasFunc

- nb_midasMapToLinear
- nb_missingEstimator.arMethod
- nb_missingEstimator.checkForMissing
- nb_missingEstimator.estimate
- nb_missingEstimator.forecastMethod
- nb_missingEstimator.getVariables
- nb_mlEstimator.bootstrapModel
- nb_mlEstimator.estimate
- nb_mlEstimator.getInitMFVAR
- nb_mlEstimator.getPredicted
- nb_mlEstimator.help
- nb_mlEstimator.makeArtificial
- nb_mlEstimator.print
- nb_mlEstimator.translateOptimization
- nb_mlarima
- nb_nls
- nb_nlsEstimator.doTest
- nb_nlsEstimator.getCoeff
- nb_nlsEstimator.getPredicted
- nb_nlsEstimator.help
- nb_nlsEstimator.print
- nb_nwhishart
- nb_ols
- nb_olsEstimator.addLeads
- nb_olsEstimator.bootstrapModel
- nb_olsEstimator.estimate
- nb_olsEstimator.getDependent
- nb_midasResiduals
- nb_missingEstimator.bootstrapModel
- nb_missingEstimator.copulaMethod
- nb_missingEstimator.fillInForMissing
- nb_missingEstimator.getStateSpace
- nb_missingEstimator.kalmanMethod
- nb_mlEstimator.drawParameters
- nb_mlEstimator.getDependent
- nb_mlEstimator.getMeasurementEqMFVAR
- nb_mlEstimator.getResidual
- nb_mlEstimator.interpretMeasurementError
- nb_mlEstimator.mfvarEstimator
- nb_mlEstimator.template
- nb_mlEstimator.varEstimator
- nb_neweyWestRobustSTD
- nb_nlsEstimator.bootstrapModel
- nb_nlsEstimator.estimate
- nb_nlsEstimator.getDependent
- nb_nlsEstimator.getResidual
- nb_nlsEstimator.makeArtificial
- nb_nlsEstimator.template
- nb_nwhishartPrior
- nb_olsEstimator.addLags
- nb_olsEstimator.addSeasonalDummies
- nb_olsEstimator.doTest
- nb_olsEstimator.getCoeff
- nb_olsEstimator.getPredicted

- nb_olsEstimator.getResidual
- nb_olsEstimator.makeArtificial
- nb_olsEstimator.print
- nb_olsEstimator.template
- nb_olsRestricted
- nb_pcaEstimator.estimate
- nb_pcaEstimator.getCoeff
- nb_pcaEstimator.getObservables
- nb_pcaEstimator.print
- nb_qreg
- nb_quantileEstimator.closeWaitbar
- nb_quantileEstimator.getCoeff
- nb_quantileEstimator.getPredicted
- nb_quantileEstimator.help
- nb_quantileEstimator.notifyWaitbar
- nb_quantileEstimator.openWaitbar
- nb_quantileEstimator.template
- nb_realTimeEstimator.estimate
- nb_ridge
- nb_ridgeEstimator.estimate
- nb_ridgeEstimator.getDependent
- nb_ridgeEstimator.getResidual
- nb_ridgeEstimator.makeArtificial
- nb_ridgeEstimator.template
- nb_rwEstimator.estimate
- nb_rwEstimator.getDependent
- nb_rwEstimator.getResidual
- nb_olsEstimator.help
- nb_olsEstimator.modelSelectionAlgorithm
- nb_olsEstimator.secureAllLags
- nb_olsEstimator.varModifications
- nb_pca
- nb_pcaEstimator.estimateFactors
- nb_pcaEstimator.getFactors
- nb_pcaEstimator.help
- nb_pcaEstimator.template
- nb_quantileEstimator.bootstrapModel
- nb_quantileEstimator.estimate
- nb_quantileEstimator.getDependent
- nb_quantileEstimator.getResidual
- nb_quantileEstimator.makeArtificial
- nb_quantileEstimator.notifyWaitbarQuantile
- nb_quantileEstimator.print
- nb_quantileEstimator.varModifications
- nb_realTimeEstimator.mergeResults
- nb_ridgeEstimator.bootstrapModel
- nb_ridgeEstimator.getCoeff
- nb_ridgeEstimator.getPredicted
- nb_ridgeEstimator.help
- nb_ridgeEstimator.print
- nb_rwEstimator.bootstrapModel
- nb_rwEstimator.getCoeff
- nb_rwEstimator.getPredicted
- nb_rwEstimator.help

- nb_rwEstimator.makeArtificial
- nb_rwEstimator.template
- nb_seasonalEstimator.estimate
- nb_seasonalEstimator.getCoeff
- nb_seasonalEstimator.getSeasonallyAdjusted
- nb_seasonalEstimator.print
- nb_setUpForDiffuseFilter
- nb_shorteningEstimator.doShortening
- nb_shorteningEstimator.getCoeff
- nb_shorteningEstimator.getDependent
- nb_shorteningEstimator.print
- nb_statespaceEstimator.checkCalibration
- nb_statespaceEstimator.getBounds
- nb_statespaceEstimator.getDependent
- nb_statespaceEstimator.getResidual
- nb_statespaceEstimator.loadData
- nb_statespaceEstimator.sampler
- nb_tsls
- nb_tslsEstimator.estimate
- nb_tslsEstimator.getDependent
- nb_tslsEstimator.getResidual
- nb_tslsEstimator.print
- nb_tvpmsfvEstimator.bootstrapModel
- nb_tvpmsfvEstimator.f_KFS_fac
- nb_tvpmsfvEstimator.f_em_mf
- nb_tvpmsfvEstimator.f_smoothed_fac_shocks
- nb_tvpmsfvEstimator.getBeta
- nb_rwEstimator.print
- nb_saveDraws
- nb_seasonalEstimator.estimateSeasonal
- nb_seasonalEstimator.getDependent
- nb_seasonalEstimator.help
- nb_seasonalEstimator.template
- nb_shorteningEstimator.doFunc
- nb_shorteningEstimator.estimate
- nb_shorteningEstimator.getData
- nb_shorteningEstimator.help
- nb_shorteningEstimator.template
- nb_statespaceEstimator.estimate
- nb_statespaceEstimator.getCoeff
- nb_statespaceEstimator.getPredicted
- nb_statespaceEstimator.help
- nb_statespaceEstimator.print
- nb_statespaceEstimator.template
- nb_tslsEstimator.bootstrapModel
- nb_tslsEstimator.getCoeff
- nb_tslsEstimator.getPredicted
- nb_tslsEstimator.help
- nb_tslsEstimator.template
- nb_tvpmsfvEstimator.estimate
- nb_tvpmsfvEstimator.f_KFS_params
- nb_tvpmsfvEstimator.f_factor_minnesotastruc
- nb_tvpmsfvEstimator.getAggregationScheme
- nb_tvpmsfvEstimator.getCoeff

- nb_tvpmsvEstimator.getDependent
- nb_tvpmsvEstimator.getNumberOfCoeff
- nb_tvpmsvEstimator.getPredicted
- nb_tvpmsvEstimator.getResidual
- nb_tvpmsvEstimator.help
- nb_tvpmsvEstimator.normalEstimation
- nb_tvpmsvEstimator.recursiveEstimation
- nb_varStateSpace
- nb_whiten
- nb_whitenEstimator.estimateFactors
- nb_whitenEstimator.getFactors
- nb_whitenEstimator.help
- nb_whitenEstimator.template
- nb_tvpmsvEstimator.getFactors
- nb_tvpmsvEstimator.getObservables
- nb_tvpmsvEstimator.getPriors
- nb_tvpmsvEstimator.getStateNames
- nb_tvpmsvEstimator.initialize
- nb_tvpmsvEstimator.print
- nb_tvpmsvEstimator.template
- nb_whiteRobustSTD
- nb_whitenEstimator.estimate
- nb_whitenEstimator.getCoeff
- nb_whitenEstimator.getObservables
- nb_whitenEstimator.print
- nb_zeroSpectrumEstimation

► **nb_lasso.beta0 ↑**

```
help = nb_lasso.beta0()
```

Description:

Get help on the beta0 option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_lasso.display ↑**

```
help = nb_lasso.display()
```

Description:

Get help on the display option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_lasso.displayer** ↑

```
help = nb_lasso.displayer()
```

Description:

Get help on the displayer option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_lasso.maxIter** ↑

```
help = nb_lasso.maxIter()
```

Description:

Get help on the maxIter option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_lasso.mode** ↑

```
help = nb_lasso.mode()
```

Description:

Get help on the mode option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_lasso.optimset](#)** ↑

```
options = nb_lasso.optimset(varargin)
```

Description:

Get optimization settings or help for use by the nb_lasso function.

Optional input:

- Run without inputs to get defaults.
- Run with 'list' as the first input to get a cellstr with the supported options.
- Run with 'help' to get a char with help on each option.
- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:

[nb_lasso](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_lasso.threshold](#)** ↑

```
help = nb_lasso.threshold()
```

Description:

Get help on the threshold option.

See also:

[nb_lasso.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► nb_arimaEstimator.bootstrapModel ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_arimaEstimator.bootstrapModel(...  
    model,options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► nb_arimaEstimator.drawParameters ↑

```
[betaDraws,sigmaDraws] = nb_arimaEstimator.drawParameters(results,...  
    options,draws,iter)
```

Description:

Draw parameters using asymptotic normality assumption and numerically calculated Hessian from ML estimation. Do not draw from the distribution of the std of the residual.

Written by Kenneth Sæterhagen Paulsen

► nb_arimaEstimator.estimate ↑

```
[results,options] = nb_arimaEstimator.estimate(options)
```

Description:

Estimate a ARIMA model with a selected algorithm.

Input:

- options : A struct on the format given by nb_arimaEstimator.template.
See also nb_arimaEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options.

See also:

```
nb_arimaEstimator.print, nb_arimaEstimator.help  
nb_arimaEstimator.template
```

Written by Kenneth Sæterhagen Paulsen

► **nb_arimaEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_arimaEstimator.getCoeff(options)
```

Description:

Get names of arima coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_arimaEstimator.getDependent** ↑

```
residual = nb_arimaEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_arimaEstimator.getMeasurementEqRes** ↑

```
u = nb_arimaEstimator.getMeasurementEqRes(options,sq,sp,beta,y,z,x)
```

Description:

Get residual from the measurement equation.

Written by Kenneth Sæterhagen Paulsen

► **nb_arimaEstimator.getPredicted** ↑

```
residual = nb_arimaEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_arimaEstimator.getResidual ↑**

```
residual = nb_arimaEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_arimaEstimator.help ↑**

```
helpText = nb_arimaEstimator.help
helpText = nb_arimaEstimator.help(option)
helpText = nb_arimaEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_arimaEstimator.makeArtificial ↑**

```
[YDRAW,start,finish,indY] = nb_arimaEstimator.makeArtificial(model, ...
options,results,method,draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► nb_arimaEstimator.print ↑

```
res = nb_arimaEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_arimaEstimator.estimate function.
- options : A struct with the estimation options from the nb_arimaEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► nb_arimaEstimator.template ↑

```
options = nb_arimaEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_arimaEstimator class constructor.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.CCCMSampler** ↑

```
[PAI,lambda,tau] = nb_bVarEstimator.CCCMSampler(Y,X,N,K,~,A_,sqrt_ht,  
PAI,lambda,tau)
```

Description:

Performs a draw from the conditional posterior of the VAR conditional mean coefficients by using the triangular algorithm. The triangularization achieves computation gains of order N^2 where N is the number of variables in the VAR. Algorithm developed in Carriero, Chan, Clark, and Marcellino (2021), Corrigendum to "Large Vector Autoregressions with stochastic volatility and non-conjugate priors."

The model is:

```
Y(t) = Pai(L)Y(t-1) + v(t); Y(t) is Nx1; t=1,...,T, L=1,...,p.  
v(t) = inv(A)*(LAMBDA(t)^0.5)*e(t); e(t) ~ N(0,I);
```

$$A = \begin{pmatrix} - & 1 & 0 & 0 & \dots & 0 \\ | & A(2,1) & 1 & 0 & \dots & 0 \\ | & A(3,1) & A(3,2) & 1 & \dots & 0 \\ | & \dots & \dots & \dots & \dots & \dots \\ | & A(N,1) & \dots & \dots & A(N,N-1) & 1 \end{pmatrix}$$

```
Lambda(t)^0.5 = diag[sqrt_h(1,t) , ..., sqrt_h(N,t)];
```

Input:

- Y : (TxN) matrix of data appearing on the LHS of the VAR
- X : (TxK) matrix of data appearing on the RHS of the VAR. The matrix needs to be ordered as: [1, y(t-1), y(t-2), ..., y(t-p)]
- N : scalar, #of variables in VAR
- K : scalar, #of regressors (=N*p+1)
- T : scalar, #of observations
- invA_ : (NxN) inverse of lower triangular covariance matrix A
- sqrt_ht : (TxN) time series of diagonal elements of volatility matrix. For a homoskedastic system, with Sigma the error variance, one can perform the LDL decomposition (command [L,D] =LDL(Sigma)) and set inv_A=L and sqrt_ht = repmat(sqrt(diag(D)'),T,1).

Output:

- One draw from (PAI|A,Lambda,data). PAI=[Pai(0), Pai(1), ..., Pai(p)].

See also:

`nb_bVarEstimator.horseshoe, nb_bVarEstimator.horseshoePrior`

Written by Carriero, Chan, Clark, and Marcellino (2021)
Adjusted by Maximilian Schröder

► **`nb_bVarEstimator.applyDummyPrior`** ↑

```
[y,X,constant,options] = nb_bVarEstimator.applyDummyPrior(...  
    options,y,X,yFull,XFull)
```

Description:

Add dummy observation prior.

See also:

`nb_bVarEstimator.doBayesian`

Written by Kenneth Sæterhagen Paulsen

► **`nb_bVarEstimator.applyMeasurementEqRestriction`** ↑

```
[H,y,mixing] = nb_bVarEstimator.applyMeasurementEqRestriction(H,y,...  
    options,mixing)
```

Description:

Apply measurement equation restrictions.

See also:

`nb_var.setMeasurementEqRestriction, nb_bVarEstimator.estimate`
`nb_bVarEstimator.recursiveEstimation`

Written by Kenneth Sæterhagen Paulsen

► **`nb_bVarEstimator.calculateMarginalLikelihood`** ↑

```
fh = nb_bVarEstimator.calculateMarginalLikelihood(par,paramMin,...  
    paramMax,hyperParam,nCoeff,y,X,yFull,XFull,nLags,options)
```

See also:

`nb_bVarEstimator.doEmpiricalBayesian`

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.calculateRMSE ↑**

```
rmse = nb_bVarEstimator.calculateRMSE(par,paramMin,paramMax,...  
hyperParam,nCoeff,y,X,yFull,XFull,nLags,options)
```

See also:

[nb_bVarEstimator.doEmpiricalBayesian](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.closeWaitbar ↑**

```
nb_bVarEstimator.closeWaitbar(h)
```

Description:

Close waitbar for producing posterior draws.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.constructInvWeights ↑**

```
invWeights = nb_bVarEstimator.constructInvWeights(prior,T)
```

Description:

Construct inverse weights used when stochastic-volatility-dummy prior is selected.

See also:

[nb_bVarEstimator.applyDummyPrior](#), [nb_bVarEstimator.glp](#)
[nb_bVarEstimator.nwishart](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.defaultHyperLearningSettings ↑**

```
settings = nb_bVarEstimator.defaultHyperLearningSettings()
```

See also:

[nb_bVarEstimator.calculateRMSE](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.doBayesian** ↑

```
[betaD,sigmaD,XX,posterior,pY] = nb_bVarEstimator.doBayesian(...  
    options,h,nLags,restrictions,y,X,yFull,XFull)  
[betaD,sigmaD,XX,posterior,pY] = nb_bVarEstimator.doBayesian(...  
    options,h,nLags,restrictions,y,X,yFull,XFull,obsSVD)
```

Description:

Estimate B-VAR model with a given prior.

See also:

[nb_bVarEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.doBayesianMF** ↑

```
[betaD,sigmaD,R,ys,XX,posterior] = nb_bVarEstimator.doBayesianMF(...  
    options,h,nLags,restrictions,y,X,freq,H,mixing)  
[betaD,sigmaD,R,ys,XX,posterior] = nb_bVarEstimator.doBayesianMF(...  
    options,h,nLags,restrictions,y,X,freq,H,mixing,obssSVD)
```

Description:

Estimate B-VAR model with a given prior.

See also:

[nb_bVarEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.doEmpiricalBayesian** ↑

```
[betaD,sigmaD,XX,posterior,options,fVal,pY] = ...  
    nb_bVarEstimator.doEmpiricalBayesian(options,h,nLags,restrictions,...  
        y,X,yFull,XFull,func)
```

Description:

Estimate B-VAR model with optimizing hyperparameters of the priors. See Giannone, Lenza and Primiceri (2014), "Prior selection for vector autoregressions"

Caution: We do not use MCMC methods to sample from the posterior of the hyperparameters even if options.hyperprior is set to true.

Written by Kenneth Sæterhagen Paulsen

► nb_bVarEstimator.doEmpiricalBayesianMF ↑

```
[betaD,sigmaD,R,ys,XX,posterior,options,fVal,pY] = ...
nb_bVarEstimator.doEmpiricalBayesian(options,h,nLags,restrictions, ...
y,X,freq,H,mixing,func)
```

Description:

Estimate B-VAR model with optimizing hyperparameters of the priors. See Giannone, Lenza and Primiceri (2014), "Prior selection for vector autoregressions"

Caution: We do not use MCMC methods to sample from the posterior of the hyperparameters even if options.hyperprior is set to true.

We do the optimization step over a balanced dataset, and then use the hyperparameters from this step to estimate the model on the full dataset.

Caution: Only missing observations at the end of the example is supported!

Written by Kenneth Sæterhagen Paulsen

► nb_bVarEstimator.dsge ↑

```
beta = nb_bVarEstimator.dsge(draws,y,x,nLags, ...
constant,timeTrend,prior,restrictions,waitbar)

[beta,sigma,X,posterior,pY] = nb_bVarEstimator.dsge(draws,y,x,nLags, ...
constant,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using dsge-prior, as in Del Negro and Schorfheide (2004), "Priors from General Equilibrium Models for VARS".

Input:

```

- y : A double matrix of size nobs x neq of the dependent
      variable of the regression(s).

- x : A double matrix of size nobs x h + neq*nlags. Where h
      is the exogenous variables of the model.

- nLags : Number of lags of the VAR. As an integer

- constant : If a constant is wanted in the estimation. Will be
      added first in the right hand side variables.

- timeTrend : If a linear time trend is wanted in the estimation.
      Will be added first/second in the right hand side
      variables. (First if constant is not given, else
      second)

- prior : A struct with the prior options. See
      nb_var.priorTemplate for more on this input

- restrictions : Index of parameters that are restricted to zero.

- waitbar : true, false or an object of class nb_waitbar5.

```

Output:

```

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the
      posterior draws of the estimated coefficients.

      Caution : If nargout == 1, this output is the marginal
      likelihood.

- sigma : A neq x neq x draws matrix with the posterior variance of
      the estimated coefficients.

- X : Regressors including constant and time-trend.

- posterior : A struct with all the needed information to do posterior
      draws.

- pY : Log marginal likelihood.

```

See also

`nb_bVarEstimator.estimate`, `nb_var`, `nb_bVarEstimator.logMarginalLikelihood`

Written by Kenneth S. Paulsen

► **`nb_bVarEstimator.dummyNormalizationConstant` ↑**

```
norm = nb_bVarEstimator.dummyNormalizationConstant(yd, xd, d, priorBeta, ...
                                                     PSI, Omega)
```

Description:

Calculate the normalization constant due dummy observation prior.

Input:

- T : Number of observation.
- n : Number of dependent variables of the VAR.
- p : Number of lags in the VAR.
- d : Number of degrees of freedom of the prior on Sigma.
- xx : Calculated as $x'x$, where x is the regressors of the VAR.
- PSI : Prior of Sigma.
- Omega : Scale matrix of the prior of beta.

Output:

- norm : Normalization constant.

Written by Kenneth Sæterhagen Paulsen

► nb_bVarEstimator.estimate ↑

```
[results,options] = nb_bVarEstimator.estimate(options)
```

Description:

This code is based on code from a paper by Koop and Korobilis (2010), Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.

Estimate a model with bayesian methods. The following priors are supported (Can be set by the prior field of the options struct):

- 'jeffrey' : Diffuse (M-C Integration)
- 'minnesota' : Minnesota (M-C Integration) or (Gibbs sampler)
- 'nwishart' : Normal-Wishart (M-C Integration)
- 'inwishart' : Independent Normal-Wishart (Gibbs sampler)
- 'glp' : This is the prior used in the paper by Giannone, Lenza and Primiceri (2014), which is of the Normal-Wishart type prior. (Gibbs sampler)
- 'dsge' : Normal-Wishart type prior with dummy observations from another model (often dsge, therefor the name).

The code is also extended to handle missing observations and mixed-frequency VAR models. Then the following priors are supported:

- 'minnesotaMF' : Minnesota (Gibbs sampler)
- 'nwishartMF' : Normal-Wishart (Gibbs sampler)
- 'inwishartMF' : Independent Normal-Wishart (Gibbs sampler)
- 'glpMF' : This is the prior used in the paper by Giannone, Lenza and Primiceri (2014), which is of

the Normal-Wishart type prior. (Gibbs sampler)

A Kalman smoother is ran to get the posterior distribution of the missing observations.

Input:

- options : A struct on the format given by nb_bVarEstimator.template.
See also nb_bVarEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

See also:

[nb_bVarEstimator.print](#), [nb_bVarEstimator.help](#), [nb_bVarEstimator.template](#)
[nb_var](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_bVarEstimator.estimateARLowFreq ↑**

[AR,sigma,ys,us] = nb_bVarEstimator.estimateARLowFreq(y,H)

Description:

Estimate AR(1) model of a series with higher frequency than the observed data using a kalman filter approach.

See also:

[nb_mlEstimator.minnesotaMF](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_bVarEstimator.expandZR ↑**

beta = nb_bVarEstimator.expandZR(beta,indZR)

Description:

Expand coefficient matrix given zero regressors.

Written by Kenneth Sætherhagen Paulsen

► **[nb_bVarEstimator.getAllMFVariables](#)** ↑

```
[ys,allEndo,exo] = nb_bVarEstimator.getAllMFVariables(options,...  
ys,H,tempDep,mfvar)
```

Description:

Get smoothed estimates of all variables of the MF-VAR model.

See also:

[nb_bVarEstimator.estimate](#), [nb_mfvar.filter](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_bVarEstimator.getAllMFVariablesRec](#)** ↑

```
[ys,allEndo,exo] = nb_bVarEstimator.getAllMFVariablesRec(options,...  
ys,H,tempDep,ss,start)
```

Description:

Get recursive smoothed estimates of all variables of the MF-VAR model.

See also:

[nb_bVarEstimator.recursiveEstimation](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_bVarEstimator.getCoeff](#)** ↑

```
[coeff,numCoeff] = nb_bVarEstimator.getCoeff(options,type)
```

Description:

Get names of coefficients.

Written by Kenneth Sætherhagen Paulsen

► **[nb_bVarEstimator.getDependent](#)** ↑

```
residual = nb_bVarEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getDummyPriorOptions ↑**

```
dummyPriorOptions = nb_bVarEstimator.getDummyPriorOptions(nLags, ...
    prior,constant,time_trend)
```

Description:

Return a structure that contains all options used by the nb_bVarEstimator.applyDummyPrior function.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getInitAndHyperParam ↑**

```
[hyperParam,nCoeff,init,lb,ub,paramMin,paramMax] = ...
    nb_bVarEstimator.getInitAndHyperParam(options,throwErr)
```

Description:

Get hyperparameters, initial values and bounds

See also:

[nb_bVarEstimator.doEmpiricalBayesian](#), [nb_bVarEstimator.print](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getMixing ↑**

```
mixing = nb_bVarEstimator.getMixing(options)
```

Description:

Get mixing options.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getPredicted ↑**

```
residual = nb_bVarEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getResidual** ↑

```
residual = nb_bVarEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getSVDPrior** ↑

```
options = nb_bVarEstimator.getSVDPrior(options,y)
```

Description:

Get prior mode and hyper prior distributions.

See also:

[nb_bVarEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getStateSpace** ↑

```
[x0,P0,d,H,R,T,c,A,B,Q,G,obs,failed] = nb_bVarEstimator.getStateSpace(...  
alpha,sigma,nDep,nLags,nExo,restr,H,R,x)
```

Description:

Go from draws from the posterior to state-space representation of the model. Used by the Kalman filter iteration when dealing with missing observations or mixed frequency VARs.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.getStateSpaceUnstable** ↑

```
[x0,P0,Pinf0,H,R,T,A,BQ,G,obs,failed] = nb_bVarEstimator.getStateSpace(...  
    alpha,sigma,nDep,nLags,nExo,restr,H,R,x0)
```

Description:

Go from draws from the posterior to state-space representation of the model. Used by the Kalman filter iteration when dealing with missing observations or mixed frequency VARs.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.glp** ↑

```
beta = nb_bVarEstimator.glp(draws,y,x,nLags,...  
    constant,constantAR,timeTrend,prior,restrictions,waitbar)  
  
[beta,sigma,X,posterior,pY] = nb_bVarEstimator.glp(draws,y,x,nLags,...  
    constant,constantAR,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using prior used in the paper by Giannone, Lenza and Primiceri (2014), "Prior selection for vector autoregressions" and Giannone, Lenza and Primiceri (2017), "Priors for the long run".

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x h + neq*nlags. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- constantAR : Include constant in AR models that is used to set up the prior for sigma.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- prior : A struct with the prior options. See nb_var.priorTemplate for more on this input

- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
Caution : If nargout == 1, this output is the marginal likelihood.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- X : Regressors including constant and time-trend.
- posterior : A struct with all the needed information to do posterior draws.
- pY : Log marginal likelihood.

See also
nb_bVarEstimator.estimate, **nb_var**, **nb_bVarEstimator.logMarginalLikelihood**

Written by Kenneth S. Paulsen

► **nb_bVarEstimator.glpMF ↑**

```
[beta,sigma,R,yM,X,posterior,prior] = nb_bVarEstimator.glpMF(draws,y,x, ...
    nLags,constant,constantAR,timeTrend,prior,restrictions,waitbar, ...
    freq,H,mixing)
```

Description:

Estimate VAR model using prior used in the paper by Giannone, Lenza and Primiceri (2014), "Prior selection for vector autoregressions" and Giannone, Lenza and Primiceri (2017), "Priors for the long run". In contrast to the **nb_bVarEstimator.glp** prior this prior also handle missing observations (but not mixed frequency VARs!).

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x h + neq*nlags. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.

```

- timeTrend : If a linear time trend is wanted in the estimation.
    Will be added first/second in the right hand side
    variables. (First if constant is not given, else
    second)

- prior : A struct with the prior options. See
    nb_var.priorTemplate for more on this input

- restrictions : Index of parameters that are restricted to zero.

- waitbar : true, false or an object of class nb_waitbar5.

- empirical : A struct with the options on how to do the empirical
    bayesian. Default is empty, i.e. do not do empirical
    bayesian.

- freq : A 1 x nDep int with the frequencies of the variables
    of the MF-VAR. Is empty if VAR with missing
    observations.

- H : Mapping of the measurement equation of the state-space
    representation of the missing observation VAR or MF-VAR
    model.

- mixing : A struct with the information on the variables that
    are observed with different frequencies.

    If empty, it is assumed that no variable is measured
    with more than one frequency.

```

Output:

```

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the
    posterior draws of the estimated coefficients.

- sigma : A neq x neq x draws matrix with the posterior variance of
    the estimated coefficients.

- R : A nobs x nobs double matrix with the measurement error
    covariance matrix.

- X : Regressors including constant and time-trend.

- yM : Observations on the dependent variables. Mean estimates
    of the unobserved values are returned. Including lags.

- posterior : A struct with all the needed information to do posterior
    draws.

- fVal : Initial log posterior on the full balanced sample.

- prior : Updated prior structure. I.e. the optimized
    hyperparameters will be reset if empirical bayesian is
    done.

```

See also
[nb_bVarEstimator.estimate](#), [nb_var](#), [nb_bVarEstimator.glp](#)

Written by Kenneth S. Paulsen

► nb_bVarEstimator.help ↑

```
helpText = nb_bVarEstimator.help  
helpText = nb_bVarEstimator.help(option)  
helpText = nb_bVarEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sætherhagen Paulsen

► nb_bVarEstimator.horseshoe ↑

```
[beta,sigma,X,posterior] = nb_bVarEstimator.inwishart(draws,y,x,...  
constant,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using horseshoe prior.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x neq*nlags + h. Where h is the exogenous variables of the model.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else

```

        second)

- prior      : A struct with the prior options. See
               nb_bVarEstimator.priorTemplate for more on this input

- restrictions : Index of parameters that are restricted to zero.

- waitbar     : true, false or an object of class nb_waitbar5.
```

Output:

```

- beta       : A (extra + neq*nlags + h) x neq x draws matrix with the
               posterior draws of the estimated coefficients.

- sigma      : A neq x neq x draws matrix with the posterior variance of
               the estimated coefficients.

- X          : Regressors including constant and time-trend.

- posterior  : A struct with all the needed information to do posterior
               draws.
```

See also
nb_bVarEstimator, nb_var

Written by Maximilian Schröder

► **nb_bVarEstimator.horseshoeGibbs** ↑

```
[beta,sigma] = nb_bVarEstimator.inwishartGibbs(draws,y,X,initBeta, ...
initSigma,a_prior,S_prior,v_post, ...
restrictions,thin,burn,waitbar)
```

Description:

Gibbs sampler draws of B-VAR with horseshoe prior.

Input:

See nb_bVarEstimator.inwishart

Output:

See nb_bVarEstimator.inwishart

Written by Maximilian Schröder

► **nb_bVarEstimator.horseshoePrior** ↑

```
[Beta,lambda,tau,sigma_sq] = nb_bVarEstimator.horseshoePrior(y,X,XX,...  
lambda,tau,sigma_sq,type)
```

Description:

Function to implement Horseshoe shrinkage prior (<http://faculty.chicagobooth.edu/nicholas.polson/research/papers/Horse.pdf>) in Bayesian Linear Regression.

```
Model: y=X\beta+\epsilon, \epsilon \sim N(0,\sigma^2)  
\beta_j \sim N(0,\sigma^2 \lambda_j^2 \tau^2)  
\lambda_j \sim Half-Cauchy(0,1), \tau \sim Half-Cauchy (0,1)  
\pi(\sigma^2) \sim 1/\sigma^2
```

This function employs the algorithm proposed in "Fast Sampling with Gaussian Scale-Mixture priors in High-Dimensional Regression" by Bhattacharya et. al. (2015). The global local scale parameters are updated via a Slice sampling scheme given in the online supplement of "The Bayesian Bridge" by Polson et. al. (2011). Two different algorithms are used to compute posterior samples of the $p \times 1$ vector of regression coefficients β . Type=1 corresponds to the proposed method in Bhattacharya et. al. and Type=2 corresponds to an algorithm provided% in Rue (2001). We recommend our algorithm when $p > n$.

Input:

- y : Response, a $n \times 1$ vector
- X : Matrix of covariates, dimension $n \times p$
- type : > 1 if sampling from posterior of beta is done by Proposed Method
> 2 if sampling is done by Rue's algorithm

Output:

- pMean : Posterior mean of Beta, a $p \times 1$ vector
- pMedian : Posterior median of Beta, a $p \times 1$ vector
- pLambda : Posterior mean of local scale parameters, a $p \times 1$ vector
- pSigma : Posterior mean of Error variance
- betaout : Posterior samples of beta

See also:

[nb_bVarEstimator.horseshoe](#), [nb_bVarEstimator.CCCMSampler](#)

Written by Antik Chakraborty (antik@stat.tamu.edu) and Anirban Bhattacharya (anirbanb@stat.tamu.edu)

Edited by Kenneth Sæterhagen Paulsen

- Made it into a function in the nb_bVarEstimator package
- Updated doc to fit the NB toolbox format.

► **nb_bVarEstimator.interpretRScale** ↑

```
options = nb_bVarEstimator.interpretRScale(options)
```

Description:

Get indicies of priors on the measurement error.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.inwishart** ↑

```
[beta,sigma,X,posterior] = nb_bVarEstimator.inwishart(draws,y,x,...  
constant,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using independent Normal-Wishart prior.

This code is based on the paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.
See page 12-13.

See also the DAG.pdf documentation.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x neq*nlags + h. Where h is the exogenous variables of the model.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- prior : A struct with the prior options. See nb_bVarEstimator.priorTemplate for more on this input
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.

Output:

```
- beta      : A (extra + neq*nlags + h) x neq x draws matrix with the
              posterior draws of the estimated coefficients.

- sigma     : A neq x neq x draws matrix with the posterior variance of
              the estimated coefficients.

- X         : Regressors including constant and time-trend.

- posterior : A struct with all the needed information to do posterior
              draws.
```

See also
`nb_bVarEstimator`, `nb_var`

Written by Kenneth S. Paulsen

► **`nb_bVarEstimator.inwishartGibbs`** ↑

```
[beta,sigma] = nb_bVarEstimator.inwishartGibbs(draws,y,X,initBeta, ...
                                                initSigma,a_prior,V_prior,S_prior,v_post, ...
                                                restrictions,thin,burn,waitbar)
```

Description:

Gibbs sampler draws of B-VAR with independent Normal-Wishart prior.

This code is based the paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.
See page 12-13.

See also the DAG.pdf documentation.

Input:

See `nb_bVarEstimator.inwishart`

Output:

See `nb_bVarEstimator.inwishart`

Written by Kenneth Sætherhagen Paulsen

► **`nb_bVarEstimator.inwishartMF`** ↑

```
beta,sigma,R,yM,X,posterior] = nb_bVarEstimator.inwishartMF(draws,y,x, ...
                                                               nLags,constant,timeTrend,prior,restrictions, ...
                                                               waitbar,H,mixing)
```

Description:

This method extends nb_bVarEstimator.inwishart so it also handle missing observations and mixed frequency VAR models.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x h. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- prior : A struct with the prior options. See nb_bVarEstimator.priorTemplate for more on this input
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.
- H : Mapping of the measurement equation of the state-space representation of the missing observation VAR or MF-VAR model.
- mixing : A struct with the information on the variables that are observed with different frequencies.
If empty, it is assumed that no variable is measured with more than one frequency.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- R : A nobs x nobs double matrix with the measurement error covariance matrix.
- yM : Observations on the dependent variables. Mean estimates of the unobserved values are returned. Including lags.
- X : Regressors including constant and time-trend. The mean of the unobserved values is returned.
- posterior : A struct with all the needed information to do posterior draws.

See also
nb_bVarEstimator.inwishart, nb_var

Written by Kenneth S. Paulsen

► **nb_bVarEstimator.inwishartMFGibbs** ↑

```
[beta,sigma,yDraws,PDraws] = nb_bVarEstimator.inwishartMFGibbs(draws,...  
y,x,H,R_prior,initBeta,initSigma,a_prior,V_prior,S_prior,...  
v_post,restrictions,thin,burn,waitbar,maxTries)
```

Description:

Gibbs sampling from posterior of a mixed frequency or missing observation B-VAR that implements a independent normal-wishart type prior. The normal part is set to zero. This method extends nb_bVarEstimator.inwishartGibbs.

Input:

See nb_bVarEstimator.inwishart

Output:

See nb_bVarEstimator.inwishart

Written by Kenneth Sætherhagen Paulsen

► **nb_bVarEstimator.jeffrey** ↑

```
[beta,sigma,x] = nb_bVarEstimator.jeffrey(draws,y,x,constant,...  
timeTrend,restrictions,waitbar)
```

Description:

Estimate VAR model using diffuse (Jeffrey) prior.

This code is based the paper by Koop and Korobilis (2009), Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.

See also the DAG.pdf documentation.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x neq*nlags + h. Where h is the exogenous variables of the model.

- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- X : Regressors including constant and time-trend.
- posterior : A struct with all the needed information to do posterior draws.

See also
nb_bVarEstimator, nb_var

Written by Kenneth S. Paulsen

► **nb_bVarEstimator.jeffreyMCI ↑**

```
[beta,sigma] = nb_bVarEstimator.jeffreyMCI(draws,X,T,numCoeff,nEq, ...
initSigma,SSE,a_ols,restrictions,waitbar)
```

Description:

Monte carlo integration of B-VAR with diffuse (Jeffery) prior.

This code is based on the paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.

See also the DAG.pdf documentation.

Input:

See **nb_bVarEstimator.jeffrey**

Output:

See **nb_bVarEstimator.jeffrey**

Written by Kenneth Sætherhagen Paulsen

► **nb_bVarEstimator.laplace** ↑

```
[beta,sigma,x] = nb_bVarEstimator.laplace(draws,y,x,constant,...  
timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using laplace prior.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x neq*nlags + h. Where h is the exogenous variables of the model.
- draws : Number of draws.
- recDraws : Number of draws to discard in the beginning.
- Xt1 : Value of predictors at time T+1.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.
- prior : Struct with prior options. See nb_var.priorTemplate.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- X : Regressors including constant and time-trend.
- posterior : A struct with all the needed information to do posterior draws.

See also
nb_bVarEstimator, nb_var

Written by Atle Loneland

► **nb_bVarEstimator.laplaceDensity** ↑

```
[beta,sigma] = nb_bVarEstimator.laplaceDensity(y,x,initBeta,initSigma,  
lam2Prior, lam2, draws, burn, thin, waitbar)
```

Description:

Draw from the posterior when using a laplace prior.

Written by Atle Loneland

► **nb_bVarEstimator.logMarginalLikelihood** ↑

```
pY = nb_bVarEstimator.logMarginalLikelihood(T,n,d,xx,A_prior,A_post,...  
PSI,Omega,eps)
```

Description:

Calculate the marginal likelihood of a B-VAR model with a normal-whishart prior. See Appendix A of Giannone, Lenza and Primiceri (2014), "Prior selection for vector autoregressions".

Input:

- T : Number of observation.
- n : Number of dependent variables of the VAR.
- d : Number of degrees of freedom of the prior on Sigma.
- A_prior : Prior coefficients.
- A_post : Posterior coefficients.
- xx : Calculated as $x'x$, where x is the regressors of the VAR.
- PSI : Prior of Sigma.
- Omega : Scale matrix of the prior of beta.
- eps : Residuals at posterior

Output:

- pY : Log marginal likelihood.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.minnesota** ↑

```
[beta,sigma,X] = nb_bVarEstimator.minnesota(draws,y,x,nLags,constant,...  
constantAR,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using minnesota prior.

This code is based on the paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.
See page 7 of Koop and Korobilis (2009).

See also the documentation DAG.pdf.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x h + neq*nlags. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- constantAR : Include constant in AR models that is used to set up the prior for sigma.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- prior : A struct with the prior options. See nb_bVarEstimator.priorTemplate for more on this input
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- X : Regressors including constant and time-trend.
- posterior : A struct with all the needed information to do posterior draws.

See also
nb_bVarEstimator, nb_var

Written by Kenneth S. Paulsen

► **[nb_bVarEstimator.minnesotaGibbs](#)** ↑

```
[beta,sigma] = nb_bVarEstimator.minnesotaGibbs(draws,y,X,initBeta,...  
initSigma,a_prior,V_prior,S_prior,v_post,restrictions,...  
burn,thin,waitbar)
```

Description:

Gibbs sampling from posterior of a B-VAR that implements a independent normal-wishart type prior. The normal part is taken from a minnesota style prior on the coefficents of the dynamics.

Input:

See `nb_bVarEstimator.minnesota`

Output:

See `nb_bVarEstimator.minnesota`

Written by Kenneth Sæterhagen Paulsen

► **[nb_bVarEstimator.minnesotaMCI](#)** ↑

```
[beta,sigma] = nb_bVarEstimator.minnesotaMCI(draws,y,X,initBeta,...  
initSigma,a_prior,V_prior,restrictions,waitbar)
```

Description:

Monte carlo integration of B-VAR with minnesota prior.

This code is based on the paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.
See page 7 of Koop and Korobilis (2009)

Input:

See `nb_bVarEstimator.minnesota`

Output:

See `nb_bVarEstimator.minnesota`

Written by Kenneth Sæterhagen Paulsen

► **[nb_bVarEstimator.minnesotaMF](#)** ↑

```
[beta,sigma,R,yM,X,posterior] = nb_bVarEstimator.minnesotaMF(draws,y,x, ...
    nLags,constant,timeTrend,prior,restrictions, ...
    waitbar,freq,H,mixing)
```

Description:

Estimate mixed frequency or missing observations B-VAR model using minnesota type prior. This method extends `nb_bVarEstimator.minnesota`.

Input:

- `y` : A double matrix of size `nobs` x `neq` of the dependent variable of the regression(s).
- `x` : A double matrix of size `nobs` x `h`. Where `h` is the exogenous variables of the model.
- `nLags` : Number of lags of the VAR. As an integer
- `constant` : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- `timeTrend` : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- `prior` : A struct with the prior options. See `nb_bVarEstimator.priorTemplate` for more on this input
- `restrictions` : Index of parameters that are restricted to zero.
- `waitbar` : `true`, `false` or an object of class `nb_waitbar5`.
- `freq` : A `1` x `nDep` int with the frequencies of the variables of the MF-VAR. Is empty if VAR with missing observations.
- `H` : Mapping of the measurement equation of the state-space representation of the missing observation VAR or MF-VAR model.
- `mixing` : A struct with the information on the variables that are observed with different frequencies.
If empty, it is assumed that no variable is measured with more than one frequency.

Output:

- `beta` : A `(extra + neq*nlags + h) x neq x draws` matrix with the posterior draws of the estimated coefficients.
- `sigma` : A `neq x neq x draws` matrix with the posterior variance of the estimated coefficients.
- `R` : A `nobs x nobs` double matrix with the measurement error

covariance matrix.

- yM : Observations on the dependent variables. Mean estimates of the unobserved values are returned. Including lags.
- X : Regressors including constant and time-trend. The mean of the unobserved values is returned.
- posterior : A struct with all the needed information to do posterior draws.

See also

`nb_bVarEstimator`, `nb_var`, `nb_mfvar`

Written by Kenneth S. Paulsen

► `nb_bVarEstimator.minnesotaMFGibbs` ↑

```
[beta,sigma,yDraws,PDraws] = nb_bVarEstimator.minnesotaMFGibbs(draws,...  
y,x,H,R_prior,initBeta,initSigma,a_prior,V_prior,S_prior,v_post,...  
restrictions,burn,thin,waitbar,maxTries,dummyPriorOptions)
```

Description:

Gibbs sampling from posterior of a mixed frequency or missing observation B-VAR that implements a independent normal-wishart type prior. The normal part is taken from a minnesota style prior on the coefficents of the dynamics. This method extends `nb_bVarEstimator.minnesotaGibbs`

Input:

See `nb_bVarEstimator.minnesotaMF`

Output:

See `nb_bVarEstimator.minnesotaMF`

Written by Kenneth Sæterhagen Paulsen

► `nb_bVarEstimator.minnesotaMFGibbs2` ↑

```
[beta,sigma,yDraws] = nb_bVarEstimator.minnesotaMFGibbs2(draws,y,x,H,...  
R_prior,initBeta,initSigma,a_prior,V_prior,...  
restrictions,burn,thin,waitbar,maxTries,dummyPriorOptions)
```

Description:

Gibbs sampling from posterior of a mixed frequency or missing observation B-VAR that implements a fixed sigma type prior. The prior for beta is from a minnesota style prior on the coefficents of the dynamics. This method extends `nb_bVarEstimator.minnesotaMCI`

Input:

See `nb_bVarEstimator.minnesotaMF`

Output:

See `nb_bVarEstimator.minnesotaMF`

Written by Kenneth Sæterhagen Paulsen

► **`nb_bVarEstimator.minnesotaVariancePrior`** ↑

```
sigma_sq = nb_bVarEstimator.minnesotaVariancePrior(prior,y,constant,...  
timeTrend,H,freq,mixing,indObservedOnly,nLagsAR)
```

Description:

Get the minnesota prior variance, when dealing with missing observations or mixed frequency data.

See also:

`nb_bVarEstimator.minnesotaMF`, `nb_bVarEstimator.glpMF`

Written by Kenneth Sæterhagen Paulsen

► **`nb_bVarEstimator.notifyWaitbar`** ↑

```
nb_bVarEstimator.notifyWaitbar(h,kk,note)
```

Description:

Notify waitbar for producing posterior draws.

Written by Kenneth Sæterhagen Paulsen

► **`nb_bVarEstimator.nwishart`** ↑

```
beta = nb_bVarEstimator.nwishart(draws,y,x,nLags,...  
constant,timeTrend,prior,restrictions,waitbar)
```

```
[beta,sigma,X,posterior,pY] = nb_bVarEstimator.nwishart(draws,y,x,nLags,...  
constant,timeTrend,prior,restrictions,waitbar)
```

Description:

Estimate VAR model using Normal-Wishart prior.

This code is based on code from a paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.

See page 9 of Koop and Korobilis (2009)

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x neq*nlags + h. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)
- prior : A struct with the prior options. See nb_bVarEstimator.priorTemplate for more on this input
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar5.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
Caution : If nargout == 1, this output is the marginal likelihood.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- X : Regressors including constant and time-trend.
- posterior : A struct with all the needed information to do posterior draws.
- pY : Log marginal likelihood.

See also
nb_bVarEstimator.estimate, nb_var, nb_bVarEstimator.logMarginalLikelihood

Written by Kenneth S. Paulsen

► nb_bVarEstimator.nwishartMCI ↑

```
[beta,sigma] = nb_bVarEstimator.nwishartMCI(draws,initBeta,initSigma,...  
a_post,V_post,S_post,v_post,restrictions,waitbar)
```

Description:

Monte carlo integration of B-VAR with Normal-Wishart prior.

This code is based on code from a paper by Koop and Korobilis (2009),
Bayesian Multivariate Time Series Methods for Empirical Macroeconomics.

See page 9 of Koop and Korobilis (2009)

Input:

See nb_bVarEstimator.nwishart

Output:

See nb_bVarEstimator.nwishart

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.nwishartMF** ↑

```
[beta,sigma,R,yM,X,posterior] = nb_bVarEstimator.nwishartMF(draws,y,x,...  
nLags,constant,timeTrend,prior,restrictions,waitbar,...  
H,mixing)
```

Description:

Estimate missing observation or mixed frequency VAR model using
a Normal-Wishart prior.

This code is extending nb_bVarEstimator.nwishart.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x h. Where h is the exogenous variables of the model.
- nLags : Number of lags of the VAR. As an integer
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)

- prior : A struct with the prior options. See nb_bVarEstimator.priorTemplate for more on this input
- restrictions : Index of parameters that are restricted to zero.
- waitbar : true, false or an object of class nb_waitbar.
- H : Mapping of the measurement equation of the state-space representation of the missing observation VAR or MF-VAR model.
- mixing : A struct with the information on the variables that are observed with different frequencies.

If empty, it is assumed that no variable is measured with more than one frequency.

Output:

- beta : A (extra + neq*nlags + h) x neq x draws matrix with the posterior draws of the estimated coefficients.
- sigma : A neq x neq x draws matrix with the posterior variance of the estimated coefficients.
- R : A nobs x 1 double matrix with the measurement error covariance matrix.
- yM : Observations on the dependent variables. Mean estimates of the unobserved values are returned. Including lags.
- X : Regressors including constant and time-trend. The mean of the unobserved values is returned.
- posterior : A struct with all the needed information to do posterior draws.

See also
[nb_bVarEstimator](#), [nb_var](#), [nb_mfvar](#)

Written by Kenneth S. Paulsen

► **[nb_bVarEstimator.nwishartMFGibbs](#)** ↑

```
[beta,sigma,yDraws,PDraws] = nb_bVarEstimator.nwishartMFGibbs(draws, ...
y,x,H,R_prior,initBeta,initSigma,a_prior,V_prior_inv,S_prior, ...
v_post,restrictions,thin,burn,waitbar,maxTries, ...
dummyPriorOptions)
```

Description:

Gibbs sampling from a missing observation or mixed frequency B-VAR model using a normal wishart prior.

Input:

See `nb_bVarEstimator.nwishart`

Output:

See `nb_bVarEstimator.nwishart`

Written by Kenneth SÃ¶terhagen Paulsen

► **`nb_bVarEstimator.openWaitbar`** ↑

```
[h,note,isWaitbar] = nb_bVarEstimator.openWaitbar(waitbar,iter)
```

Description:

Open up waitbar for producing posterior draws.

Written by Kenneth SÃ¶terhagen Paulsen

► **`nb_bVarEstimator.print`** ↑

```
res = nb_bVarEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- `results` : A struct with the estimation results from the `nb_bVarEstimator.estimate` function.
- `options` : A struct with the estimation options from the `nb_bVarEstimator.estimate` function.
- `precision` : The precision of the printed result.

Output:

- `results` : A char with the estimation results.

Written by Kenneth SÃ¶terhagen Paulsen

► **`nb_bVarEstimator.priorTemplate`** ↑

```
template = nb_bVarEstimator.priorTemplate(type)
```

Description:

Get prior template for a given BVAR prior type. Assign it to the prior field of the main estimation template.

Input:

- type : A string with the prior type. See nb_var.priorTemplate or nb_mfvar.priorTemplate for more on the output.

Output:

- template : A struct with the options of the different priors

Examples:

```
template = nb_bVarEstimator.priorTemplate('jeffrey')
```

See also:

[nb_var.priorTemplate](#), [nb_mfvar.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.recursiveEstimation** ↑

```
[res,options] = nb_bVarEstimator.recursiveEstimation(options,y,X,...  
mfvar,missing)
```

Description:

Estimate B-VAR model recursively.

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.removeZR** ↑

```
[x,indZR,restrictions] = nb_bVarEstimator.removeZR(x,constant,...  
timeTrend,numDep,nLags,restrictions)
```

Description:

Remove zero regressors.

Written by Kenneth Sæterhagen Paulsen

► **[nb_bVarEstimator.setUpPriorDIO](#)** ↑

```
[yPlus,XPlus] = nb_bVarEstimator.setUpPriorDIO(prior,y,x,lags,...  
constant,timeTrend)
```

Description:

Set up artificial series when using a dummy-initial-observation prior by Sims (1993).

See also:

[nb_bVarEstimator.applyDummyPrior](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_bVarEstimator.setUpPriorLR](#)** ↑

```
[yPlus,XPlus] = nb_bVarEstimator.setUpPriorLR(prior,y,x,lags,...  
constant,timeTrend)
```

Description:

Set up artificial series when using a prior for the long run as in Giannone et. al (2014).

See also:

[nb_bVarEstimator.applyDummyPrior](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_bVarEstimator.setUpPriorSC](#)** ↑

```
[yPlus,XPlus] = nb_bVarEstimator.setUpPriorSC(prior,y,x,lags,...  
constant,timeTrend)
```

Description:

Set up artificial series when using a sum-of-coefficients prior by Doan, Litterman, and Sims (1984).

See also:

[nb_bVarEstimator.applyDummyPrior](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.template** ↑

```
options = nb_bVarEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_bVarEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

See also:

[nb_bVarEstimator.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.testLRPriorH** ↑

```
H = nb_bVarEstimator.testLRPriorH(H,n)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_bVarEstimator.testLRPriorPhi** ↑

```
phi = nb_bVarEstimator.testLRPriorPhi(phi,n)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.bootstrapModel** ↑

```
[betaDraws,factorDraws] =
nb_dfmemlEstimator.bootstrapModel(options,res,iter)
```

Description:

Not possible to bootstrap this model.

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.checkConvergence ↑**

```
converged = nb_dfmemlEstimator.checkConvergence(res,oldLogLik,tol)
```

Description:

Check for convergence of the EML-algorithm based on minus the log likelihood.

See also:

[nb_dfmemlEstimator.normalEstimation](#)

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.emlIteration ↑**

```
results = nb_dfmemlEstimator.emlIteration(options,results,X)
```

Description:

Do one iteration of the expected maximum likelihood algorithm for estimating a dynamic factor model.

Inputs:

- options : A struct with estimation options. See [nb_dfmemlEstimator.template](#) or [nb_dfmemlEstimator.help](#).
- results : A struct with the output from [nb_dfmemlEstimator.initialize](#).
- X : A T x N double storing the data on the observables.

Output:

- results : A struct with the updated matrices and value of the likelihood at the current iteration.

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.estimate ↑**

```
[results,options] = nb_dfmemlEstimator.estimate(options)
```

Description:

Estimate a dynamic factor model with the two step expected likelihood algorithm suggested by Banbura et al. (2010). Some small alteration to the algorithm may have been done in special cases. See the help on the nb_fmdyn class for more on the algorithm implemented by this package.

Input:

- options : A struct on the format given by nb_dfmemlEstimator.template. See also nb_dfmemlEstimator.help.

Output:

- result : A struct with the estimation results.
- options : Some options may be updated.

See also:

[nb_dfmemlEstimator.print](#), [nb_dfmemlEstimator.help](#)
[nb_dfmemlEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getBQ** ↑

```
results = nb_dfmemlEstimator.getBQ(results)
```

Description:

Standardize residuals of the idiosyncratic components to have a covariance matrix as the identity matrix.

See also:

[nb_dfmemlEstimator.initialize](#), [nb_dfmemlEstimator.emlIteration](#)

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_dfmemlEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getDependent** ↑

```
dependent = nb_dfmemlEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object. There are none, see getObservables instead.

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getEpsInd** ↑

```
indEps = nb_dfmemlEstimator.getEpsInd(options)
```

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.getFactors** ↑

```
factors = nb_fmEstimator.getFactors(results,options)
```

Description:

Get the estimated factors of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getIdiosyncraticMapping** ↑

```
Hlow = nb_dfmemlEstimator.getIdiosyncraticMapping(options)
```

Description:

Get the mixed frequency mapping matrix.

Inputs:

- options : A struct with estimation options. See
nb_dfmemlEstimator.template or nb_dfmemlEstimator.help.

Written by Kenneth S. Paulsen

► **`nb_dfmemlEstimator.getMapping`** ↑

```
[R,r] = nb_dfmemlEstimator.getMapping(options,index)
```

Description:

Get the mixed frequency mapping matrix.

Inputs:

- options : A struct with estimation options. See
 `nb_dfmemlEstimator.template` or `nb_dfmemlEstimator.help`.
- index : The index of the low frequency variable of interest.

Written by Kenneth S. Paulsen

► **`nb_dfmemlEstimator.getNumberOfCoeff`** ↑

```
nCoeff = nb_dfmemlEstimator.getNumberOfCoeff(options)
```

Description:

Get the number of estimated coefficients of the dynamic factor model.

Written by Kenneth S. Paulsen

► **`nb_dfmemlEstimator.getObservables`** ↑

```
observables = nb_dfmemlEstimator.getObservables(results,options)
```

Description:

Get the observables of the estimated model as a `nb_ts` object

Written by Kenneth Sætherhagen Paulsen

► **`nb_dfmemlEstimator.getPredicted`** ↑

```
predicted = nb_dfmemlEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getResidual** ↑

```
residual = nb_dfmemlEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object. This is the idiosyncratic component of the measurement error, and not the residual of the transition equation.

Written by Kenneth Sæterhagen Paulsen

► **nb_dfmemlEstimator.getSolution** ↑

```
results = nb_dfmemlEstimator.getSolution(options,H,AF,QF,AI,QI,R)
```

Description:

We want the solution on the form of equation 12 of Banbura et al. (2010), "Nowcasting", but introduce the residual impact matrix to reduce the dimension of Q:

X(t) = Z*alpha(t) + nu(t)

alpha(t) = T*alpha(t-1) + BQ*e(t)

where e(t) ~ N(0,I) and nu(t) ~ N(0,R).

See also:

[nb_dfmemlEstimator.initialize](#), [nb_dfmemlEstimator.emlIteration](#)

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.getStateNames** ↑

```
[stateNames,resNames] = nb_dfmemlEstimator.getStateNames(options)
```

Description:

Selects the wanted model class and estimate the model.

Written by Kenneth S. Paulsen

► **`nb_dfmemlEstimator.help`** ↑

```
helpText = nb_dfmemlEstimator.help  
helpText = nb_dfmemlEstimator.help(option)  
helpText = nb_dfmemlEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sætherhagen Paulsen

► **`nb_dfmemlEstimator.initialize`** ↑

```
results = nb_dfmemlEstimator.initialize(options,X)
```

Description:

Initialization of the expected maximum likelihood estimator of the dynamic factor model.

Inputs:

- options : A struct with estimation options. See `nb_dfmemlEstimator.template` or `nb_dfmemlEstimator.help`.
- X : A T x N double storing the data on the observables.

Written by Kenneth S. Paulsen

► **`nb_dfmemlEstimator.intiKF`** ↑

```
[alpha0,P0,Pinf0] = nb_dfmemlEstimator.intiKF(results)
```

Description:

Initialize Kalman filter.

See also:

[nb_dfmemlEstimator.emlIteration](#), [nb_dfmemlEstimator.normalEstimation](#)

Written by Kenneth S. Paulsen

► **[nb_dfmemlEstimator.normalEstimation](#)** ↑

```
[res,options] = nb_dfmemlEstimator.normalEstimation(options,results)
```

Description:

Selects the wanted model class and estimate the model.

Inputs:

- options : A struct on the format returned by
`nb_dfmemlEstimator.template`.
- results : A struct with initial conditions for EML iterations. Can be used for recursive estimation. If empty or not provided this struct is provided by `nb_dfmemlEstimator.initialize`

Written by Kenneth S. Paulsen

► **[nb_dfmemlEstimator.print](#)** ↑

```
res = nb_dfmemlEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the `nb_dfmemlEstimator.estimate` function.
- options : A struct with the estimation options from the `nb_dfmemlEstimator.estimate` function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_dfmemlEstimator.recursiveEstimation** ↑

```
[results,options] = nb_dfmemlEstimator.recursiveEstimation(options)
```

Description:

Selects the wanted model class and recursively estimate the model.

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.removeLeadingAndTrailingNaN** ↑

```
options = nb_dfmemlEstimator.removeLeadingAndTrailingNaN(options)
```

Description:

Remove leading and trailing nans from dataset.

Written by Kenneth S. Paulsen

► **nb_dfmemlEstimator.template** ↑

```
options = nb_dfmemlEstimator.template()
```

Description:

Construct a struct which must be provided to the
nb_dfmemlEstimator.estimate function.

This structure provided the user the possibility to set different
estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_dynareEstimator.print** ↑

```
res = nb_dynareEstimator.print(results,options,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_dsgen object when running a Dynare model file that does estimation.
- options : A struct with the estimation options from the nb_dsgen object when running a Dynare model file that does estimation.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.addECMLags** ↑

```
[options,maxLags] = nb_ecmEstimator.addECMLags(options,fixed)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.bootstrapModel** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_ecmEstimator.bootstrapModel(...  
model,options,results,method,draws,iter)
```

Description:

Bootstrap model.

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.estimate** ↑

```
[results,options] = nb_ecmEstimator.estimate(options)
```

Description:

Estimate a ECM model with ols.

Input:

- options : A struct on the format given by nb_ecmEstimator.template.
See also nb_ecmEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

See also:

[nb_ecmEstimator.print](#), [nb_ecmEstimator.help](#), [nb_ecmEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.getCoeff** ↑

[coeff,numCoeff] = nb_ecmEstimator.getCoeff(options)

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.getDependent** ↑

dependent = nb_ecmEstimator.getDependent(results,options)

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_ecmEstimator.getPredicted** ↑

predicted = nb_ecmEstimator.getPredicted(results,options)

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ecmEstimator.getResidual** ↑

```
residual = nb_ecmEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ecmEstimator.help** ↑

```
helpText = nb_ecmEstimator.help  
helpText = nb_ecmEstimator.help(option)  
helpText = nb_ecmEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ecmEstimator.makeArtificial** ↑

```
[YDRAW,start,finish,indy,startEst] = nb_ecmEstimator.makeArtificial(...  
model,options,results,method,...  
draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► nb_ecmEstimator.print ↑

```
res = nb_ecmEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_ecmEstimator.estimate function.
- options : A struct with the estimation options from the nb_ecmEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► nb_ecmEstimator.template ↑

```
options = nb_ecmEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_ecmEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.addExoLags](#)** ↑

```
[options,maxLags] = nb_estimator.addExoLags(options,field)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.addRestrictions](#)** ↑

```
res = nb_estimator.addRestrictions(options,res,restrict)
```

See also:

[nb_olsEstimator.estimate](#), [nb_estimator.applyRestrictions](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.applyRestrictions](#)** ↑

```
[yRest,XRest,restrict] = nb_estimator.applyRestrictions(options,y,X)
```

See also:

[nb_olsEstimator.estimate](#), [nb_estimator.addRestrictions](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.checkDOF](#)** ↑

```
nb_estimator.checkDOF(options,numCoeff,T)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.checkDOFRecursive](#)** ↑

```
[start,iter,ss] = nb_estimator.checkDOFRecursive(options,numCoeff,T)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_estimator.closeWaitbar](#)** ↑

```
nb_estimator.closeWaitbar(h)
```

Description:

Close waitbar for recursive estimation.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.correctForMissing** ↑

```
y = nb_estimator.correctForMissing(y,missing)
```

Description:

Set observations to nan so each recursion uses the same ragged edge!

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.correctOptionsForUnbalanced** ↑

```
options = nb_estimator.correctOptionsForUnbalanced(options)
```

See also:

[nb_olsEstimator.estimate](#), [nb_quantileEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.correctResultsGivenUnbalanced** ↑

```
[res,options] = nb_estimator.correctResultsGivenUnbalanced(options,res)
```

Description:

Expand beta and exogeneous to be equal when exogenous variables are leaded and not.

See also:

[nb_olsEstimator.estimate](#), [nb_quantileEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.fillInForMissing** ↑

```
[X,nanInd] = nb_estimator.fillInForMissing(X)
```

Description:

Fill in for missing values.

Input:

- X : A T x N double storing the data on the observables.

Output:

- X : A T x N double storing the balanced dataset on the observables.

- nanInd : A T x N logical. true at the elements where data is missing.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.getBlockExogenousRestrictions** ↑

```
restrictions = nb_estimator.getBlockExogenousRestrictions(options)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.getBounds** ↑

```
[lbo,ubo] = nb_estimator.getBounds(prior,lb,ub)
```

Description:

This function can be used for finding the finite support for those optimizers that needs that. E.g. nb_abc and bee_gate.

Input:

- prior : A struct with the prior specification.
- lb : The lower bound, as a nParam x 1 double.
- ub : The upper bound, as a nParam x 1 double.

Output:

- lbo : The lower bound where all non-finite elements are substituted with their 1 percentile.
- ubo : The upper bound where all non-finite elements are substituted with their 99 percentile.

See also:

[nb_abc](#), [bee_gate](#), [nb_statespaceEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.getSpecificPriors** ↑

```
options = nb_estimator.getSpecificPriors(options)
```

Description:

Get indicies of priors on specific coefficients

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.getVariables** ↑

```
[vars,indV] = nb_estimator.getVariables(options)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.interpretPrecision** ↑

```
precision = nb_estimator.interpretPrecision(precision)
```

Description:

Interpret the precision input to the print function of the different estimator packages.

Input:

- precision : The precision input to the different print functions.

Output:

- precision : The interpreted precision input.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.notifyWaitbar** ↑

```
nb_estimator.notifyWaitbar(h,kk,iter,note)
```

Description:

Notify waitbar for recursive estimation.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.openWaitbar** ↑

```
[h,doDelete] = nb_estimator.openWaitbar(options,iter)
```

Description:

Open up waitbar for recursive estimation.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.print** ↑

```
res = nb_estimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from for example the nb_olsEstimator.estimate function.
- options : A struct with the estimation options from for example the nb_olsEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.printCov ↑**

```
res = nb_estimator.printCov(results,precision)
```

Description:

Get the estimated covariance matrix results as a char.

Input:

- results : A struct with the estimation results from for example the nb_olsEstimator.estimate function.
- options : A struct with the estimation options from for example the nb_olsEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sætherhagen Paulsen

► **nb_estimator.removeLeadingAndTrailingNaN ↑**

```
[X,keep] = nb_estimator.removeLeadingAndTrailingNaN(X)
```

Description:

Remove leading and trailing nans from dataset.

Written by Kenneth S. Paulsen

► **nb_estimator.set2nan ↑**

```
X = nb_estimator.set2nan(X,observables,set2nan)
```

Description:

Set observations of the data to nan.

Input:

```
- X : A T x N double.  
  
- startDate : The start date of the data in X.  
  
- observables : A 1 x N cellstr with the names of the variables.  
  
- set2nan : A struct on the format struct('VarName1',dates1,  
    'VarName2',dates2). See examples on the use of this input  
    under examples. If you use the name 'all', then the  
    data of all variables are set to nan!
```

Output:

```
- X : A T x N double.
```

Examples:

```
set2nanA = struct('all',{nb_quarter(2,2000):nb_quarter(4,2001)})  
set2nan1 = struct('VarName',{nb_quarter(2,2000):nb_quarter(4,2001)})  
set2nan2 = struct('VarName1',{nb_quarter(2,2000):nb_quarter(4,2001)},...  
    'VarName2',{nb_quarter(2,2000):nb_quarter(2,2001)})
```

See also:

[nb_tvpmfsvEstimator.normalEstimation](#), [nb_dfmemlEstimator.normalEstimation](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.template** ↑

```
options = nb_estimator.template(type)
```

Description:

Options shared by many.

Written by Kenneth Sæterhagen Paulsen

► **nb_estimator.testSample** ↑

```
[options,varargout] = nb_estimator.testSample(options,varargin)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.bootstrapModel** ↑

```
[betaDraws,sigmaDraws,options] = nb_exprEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.eqs2funcSub ↑**

```
[eqFunc,nLags,varsOut] = nb_exprEstimator.eqs2funcSub(vars,eq,type)
```

Description:

Convert equation of the model to a function handle. Subroutine.

Input:

- vars : Allowed variables in the expression.
- eq : A one line char with one equation of the model.
- type : Give 1 to indicate that you are interpreting the left hand side variables.

See also:

[nb_exprEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.estimate ↑**

```
[results,options] = nb_exprEstimator.estimate(options)
```

Description:

Estimate a model with ols.

Input:

- options : A struct on the format given by nb_exprEstimator.template.
See also nb_exprEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options.

See also:

[nb_exprEstimator.print](#), [nb_exprEstimator.help](#), [nb_exprEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_exprEstimator.getCoeff(options,eqId)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.getDependent** ↑

```
dependent = nb_exprEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.getPredicted** ↑

```
predicted = nb_exprEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.getResidual** ↑

```
residual = nb_exprEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.help** ↑

```
helpText = nb_olsEstimator.help  
helpText = nb_olsEstimator.help(option)  
helpText = nb_olsEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.makeArtificial** ↑

```
[YDRAW,X] = nb_exprEstimator.makeArtificial(model,options,results,...  
method,draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.modelSelectionAlgorithm** ↑

```
options = nb_olsEstimator.modelSelectionAlgorithm(options,minLags)
```

Description:

Function to automatically select model.

See also:

[nb_olsEstimator.estimate](#), [nb_bVarEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_exprEstimator.print](#)** ↑

```
res = nb_exprEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the `nb_exprEstimator.estimate` function.
- options : A struct with the estimation options from the `nb_exprEstimator.estimate` function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **[nb_exprEstimator.secureAllLags](#)** ↑

```
options = nb_olsEstimator.secureAllLags(options)
```

Description:

Secure that all lags are added to the data, as they are needed for forecasting, shock decomposition etc. later.

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.template** ↑

```
options = nb_olsEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_olsEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_exprEstimator.varModifications** ↑

```
options = nb_olsEstimator.varModifications(options)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.bootstrapModel** ↑

Bootstrap parameters and factors of the Factor model

See Aastveit, Gerdrup, Jore and Thorsrud (2013)

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth S. Paulsen

► **nb_fmEstimator.estimate** ↑

```
[results,options] = nb_fmEstimator.estimate(options)
```

Description:

Estimate a factor model with ols using principal components.

Input:

- options : A struct on the format given by nb_fmEstimator.template.
See also nb_fmEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

See also:

[nb_fmEstimator.print](#), [nb_fmEstimator.help](#), [nb_fmEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.estimateFactors** ↑

No documentation

Written by Kenneth S. Paulsen

► **nb_fmEstimator.getAllDynamicRegressors** ↑

[allRegressors, restrictions] = nb_fmEstimator.getAllDynamicRegressors(options,type)

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getCoeff** ↑

[coeff, numCoeff] = nb_fmEstimator.getCoeff(options, observationEq)

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getDependent** ↑

dependent = nb_fmEstimator.getDependent(results, options)

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getFactors** ↑

```
factors = nb_fmEstimator.getFactors(results,options)
```

Description:

Get the estimated factors of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getObservables** ↑

```
observables = nb_fmEstimator.getObservables(results,options)
```

Description:

Get the observables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getPredicted** ↑

```
predicted = nb_fmEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.getResidual** ↑

```
residual = nb_fmEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **[nb_fmEstimator.help](#)** ↑

```
helpText = nb_fmEstimator.help  
helpText = nb_fmEstimator.help(option)  
helpText = nb_fmEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶therhagen Paulsen

► **[nb_fmEstimator.modelSelectionAlgorithm](#)** ↑

```
options = nb_fmEstimator.modelSelectionAlgorithm(options)
```

Description:

Function to automatically select model.

See also:

[nb_olsEstimator.estimate](#), [nb_bVarEstimator.estimate](#)

Written by Kenneth SÃ¶therhagen Paulsen

► **[nb_fmEstimator.normalEstimation](#)** ↑

```
[res,options] = nb_fmEstimator.normalEstimation(options,res)
```

Description:

Selects the wanted model class and estimate the model.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.normalEstimationDynamic ↑**

```
[res,options] = nb_fmEstimator.normalEstimationDynamic(options,res)
```

Description:

Estimation algorithm of dynamic factor models with pre-estimated factors by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.normalEstimationFAVAR ↑**

```
[res,options] = nb_fmEstimator.normalEstimationFAVAR(options,res)
```

Description:

Estimation algorithm of factor augmented VAR models with pre-estimated factors by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.normalEstimationSingleEq ↑**

```
[res,options] = nb_fmEstimator.normalEstimationSingleEq(options,res)
```

Description:

Estimation algorithm of steep ahead factor models with pre-estimated factors by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.normalEstimationStepAhead ↑**

```
[res,options] = nb_fmEstimator.normalEstimationStepAhead(options,res)
```

Description:

Estimation algorithm of steep ahead factor models with pre-estimated factors by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.print** ↑

```
res = nb_fmEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_fmEstimator.estimate function.
- options : A struct with the estimation options from the nb_fmEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_fmEstimator.recursiveEstimation** ↑

```
[res,options] = nb_fmEstimator.recursiveEstimation(options,res)
```

Description:

Selects the wanted model class and recursively estimate the model.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.recursiveEstimationDynamic** ↑

```
[res,options] = nb_fmEstimator.recursiveEstimationDynamic(options,res)
```

Description:

Recursive estimation algorithm of dynamic factor models with factors estimated by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.recursiveEstimationFAVAR** ↑

```
[res,options] = nb_fmEstimator.normalEstimationFAVAR(options,res)
```

Description:

Recursive estimation algorithm of factor augmented VAR models with factors estimated by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.recursiveEstimationSingleEq** ↑

```
[res,options] = nb_fmEstimator.recursiveEstimationSingleEq(options,res)
```

Description:

Recursive estimation algorithm of steep ahead factor models with factors estimated by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.recursiveEstimationStepAhead** ↑

```
[res,options] = nb_fmEstimator.recursiveEstimationStepAhead(options,res)
```

Description:

Recursive estimation algorithm of steep ahead factor models with factors estimated by principal components.

Written by Kenneth S. Paulsen

► **nb_fmEstimator.template** ↑

```
options = nb_fmEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_fmEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_hpEstimator.estimate ↑**

[results,options] = nb_hpEstimator.estimate(options)

Description:

Estimate HP-filtered series.

Input:

- options : A struct on the format given by nb_hpEstimator.template.
See also nb_hpEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change depending on inputs.

See also:

[nb_hpEstimator.print](#), [nb_hpEstimator.help](#)
[nb_hpEstimator.template](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_hpEstimator.estimateHP ↑**

No documentation

Written by Kenneth S. Paulsen

► **nb_hpEstimator.getCoeff ↑**

```
[coeff,numCoeff] = nb_hpEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_hpEstimator.getDependent ↑**

```
dependent = nb_hpEstimator.getDependent(results,options)
```

Description:

Get the dependent variables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_hpEstimator.getHP ↑**

```
seasonal = nb_hpEstimator.getHP(results,options)
```

Description:

Get the estimated HP-filtered series as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_hpEstimator.help ↑**

```
helpText = nb_shorteningEstimator.help  
helpText = nb_shorteningEstimator.help(option)  
helpText = nb_shorteningEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_hpEstimator.print** ↑

```
res = nb_shorteningEstimator.print(results,options,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_shorteningEstimator.estimate function.
- options : A struct with the estimation options from the nb_shorteningEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_hpEstimator.template** ↑

```
options = nb_shorteningEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_shorteningEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **[nb_lassoEstimator.bootstrapModel](#)** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_lassoEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **[nb_lassoEstimator.estimate](#)** ↑

```
[results,options] = nb_lassoEstimator.estimate(options)
```

Description:

Estimate a model with LASSO.

Input:

- options : A struct on the format given by nb_lassoEstimator.template.
See also nb_lassoEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using
the 'modelSelection' options.

See also:

[nb_lassoEstimator.print](#), [nb_lassoEstimator.help](#)
[nb_lassoEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_lassoEstimator.getCoeff](#)** ↑

```
[coeff,numCoeff] = nb_lassoEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_lassoEstimator.getDependent** ↑

```
dependent = nb_lassoEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_lassoEstimator.getPredicted** ↑

```
predicted = nb_lassoEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_lassoEstimator.getResidual** ↑

```
residual = nb_lassoEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_lassoEstimator.help** ↑

```
helpText = nb_lassoEstimator.help
helpText = nb_lassoEstimator.help(option)
helpText = nb_lassoEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_lassoEstimator.makeArtificial ↑**

```
[YDRAW,start,finish,indy,startEst] = nb_lassoEstimator.makeArtificial(...  
    model,options,results,method,...  
    draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_lassoEstimator.print ↑**

```
res = nb_lassoEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_lassoEstimator.estimate function.
- options : A struct with the estimation options from the nb_lassoEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- res : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_lassoEstimator.template** ↑

```
options = nb_lassoEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_lassoEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.addLags** ↑

```
[options,freqMapping] = nb_midasEstimator.addLags(options)
```

Description:

Add lags to right hand side of estimation equation.

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.bootstrapModel** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_midasEstimator.bootstrapModel(model,...  
options,results,method,draws,iter,forceNewDraws)
```

Description:

Bootstrap MIDAS model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **[nb_midasEstimator.checkDOFRecursive](#)** ↑

```
[start,iter,ss] = nb_midasEstimator.checkDOFRecursive(options,numCoeff,T)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_midasEstimator.estimate](#)** ↑

```
[results,options] = nb_midasEstimator.estimate(options)
```

Description:

Estimate a MIDAS model with wanted algorithm.

Input:

- options : A struct on the format given by nb_midasEstimator.template.
See also nb_midasEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

See also:

[nb_midasEstimator.print](#), [nb_midasEstimator.help](#), [nb_midasEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_midasEstimator.getCoeff](#)** ↑

```
[coeff,numCoeff] = nb_midasEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **[nb_midasEstimator.getDependent](#)** ↑

```
dependent = nb_midasEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_midasEstimator.getPredicted** ↑

```
predicted = nb_midasEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_midasEstimator.getResidual** ↑

```
residual = nb_midasEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_midasEstimator.help** ↑

```
helpText = nb_midasEstimator.help  
helpText = nb_midasEstimator.help(option)  
helpText = nb_midasEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.modelSelectionAlgorithm** ↑

```
options = nb_midasEstimator.modelSelectionAlgorithm(options)
```

Description:

Function to automatically select model.

See also:

[nb_midasEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.normalEstimation** ↑

```
[results,options] = nb_midasEstimator.normalEstimation(options,y,X,nExo)
```

Description:

Estimate MIDAS model over the full sample.

See also:

[nb_midasEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.print** ↑

```
res = nb_midasEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_midasEstimator.estimate function.
- options : A struct with the estimation options from the nb_midasEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.recursiveEstimation** ↑

```
[results,options] = nb_midasEstimator.recursiveEstimation(options,...  
y,X,nExo,startLowPeriods)
```

Description:

Estimate MIDAS model recursivly.

See also:

[nb_midasEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_midasEstimator.template** ↑

```
options = nb_midasEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_midasEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **[nb_missingEstimator.arMethod](#)** ↑

```
options = nb_missingEstimator.arMethod(options)
```

Description:

Fill in for missing observations using AR models.

Written by Kenneth Sæterhagen Paulsen

► **[nb_missingEstimator.bootstrapModel](#)** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_missingEstimator.bootstrapModel(...  
model,options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **[nb_missingEstimator.checkForMissing](#)** ↑

```
options = nb_missingEstimator.checkForMissing(options,...  
startInd,endInd)
```

Description:

Check for missing observations.

Written by Kenneth Sæterhagen Paulsen

► **[nb_missingEstimator.copulaMethod](#)** ↑

```
options = nb_missingEstimator.copulaMethod(options)
```

Description:

Fill in for missing observations using a copula model.

Written by Kenneth Sæterhagen Paulsen

► **nb_missingEstimator.estimate** ↑

```
[results,options] = nb_missingEstimator.estimate(options)
```

Description:

Estimate a model with missing data (possibly with real-time data).

Input:

- options : A struct on the format given by underlying estimators template function. See for example nb_olsEstimator.template.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

Written by Kenneth Sæterhagen Paulsen

► **nb_missingEstimator.fillInForMissing** ↑

```
options = nb_missingEstimator.fillInForMissing(options,check)
```

Description:

Fill in for missing observations.

Written by Kenneth Sæterhagen Paulsen

► **nb_missingEstimator.forecastMethod** ↑

```
options = nb_missingEstimator.forecastMethod(options)
```

Description:

Fill in for missing observations using forecast with information up until time t-1, when the first missing observation is at time t.

Written by Kenneth Sæterhagen Paulsen

► **nb_missingEstimator.getStateSpace** ↑

```
[x0,P0,d,H,R,T,c,A,B,Q,G] = nb_missingEstimator.getStateSpace(...  
options,model)
```

Description:

Go from the state space representation in the model struct to the state space matrices needed by the nb_kalmansmooth_missing function.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_missingEstimator.getVariables ↑**

```
[endo,exo] = nb_missingEstimator.getVariables(options)
```

Description:

Get the variables of the model.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_missingEstimator.kalmanMethod ↑**

```
options = nb_missingEstimator.kalmanMethod(options)
```

Description:

Fill in for missing observations a Kalman filter.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_mlEstimator.bootstrapModel ↑**

```
[betaDraws,sigmaDraws,estOpt] = nb_mlEstimator.bootstrapModel(...  
model,options,results,method,draws,iter)
```

Description:

Bootstrap model.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_mlEstimator.drawParameters ↑**

```
[betaDraws,sigmaDraws,yD,pD] = nb_mlEstimator.drawParameters(results,...  
draws,iter)
```

Description:

Draw parameters using asymptotic normality assumption and numerically calculated Hessian from ML estimation. Do not draw from the distribution of the std of the residual.

If the model include missing observation these are observations are re-estimated using the Kalman filter for each draw from the confidence set of the parameters.

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.estimate ↑**

```
[results,options] = nb_mlEstimator.estimate(options)
```

Description:

Estimate a model with maximum likelihood using a Kalman filter.

Input:

- options : A struct on the format given by nb_mlEstimator.template.
See also nb_mlEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using
the 'modelSelection' options.

See also:

[nb_mlEstimator.print](#), [nb_mlEstimator.help](#), [nb_mlEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.getDependent ↑**

```
dependent = nb_olsEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.getInitMFVAR ↑**

```
[betaExo,betaDyn,sigmaPar] = nb_mlEstimator.getInitMFVAR(y,X,options,...  
method,H)
```

Description:

Get initial values for optimization routine when dealing with a MF-VAR model.

See also:

[nb_mlEstimator.mfvarEstimator](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.getMeasurementEqMFVAR ↑**

```
[H,freqD,extraAdded] = nb_mlEstimator.getMeasurementEqMFVAR(options,extra)
```

Description:

Get measurement equation of a mixed frequency VAR.

See also:

[nb_mlEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.getPredicted ↑**

```
predicted = nb_olsEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.getResidual** ↑

```
residual = nb_olsEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.help** ↑

```
helpText = nb_mlEstimator.help  
helpText = nb_mlEstimator.help(option)  
helpText = nb_mlEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.interpretMeasurementError** ↑

```
options = nb_mlEstimator.interpretMeasurementError(options)
```

Description:

Get indicies of the measurement error parameters to estimate.

Written by Kenneth Sæterhagen Paulsen

► **[nb_mlEstimator.makeArtificial](#)** ↑

```
YDRAW = nb_mlEstimator.makeArtificial(model,options,results,...  
method,draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **[nb_mlEstimator.mfvarEstimator](#)** ↑

```
[beta,stdBeta,tStatBeta,pValBeta,sigma,residual,ys,ysl,lik,Omega,pD] =  
nb_mlEstimator.mfvarEstimator(y,X,options,H,init)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_mlEstimator.print](#)** ↑

```
res = nb_olsEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_olsEstimator.estimate function.
- options : A struct with the estimation options from the nb_olsEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **[nb_mlEstimator.template](#)** ↑

```
options = nb_mlEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_mlEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.translateOptimization** ↑

```
nb_mlEstimator.translateOptimization(exitflag)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_mlEstimator.varEstimator** ↑

```
[beta, stdBeta, tStatBeta, pValBeta, sigma, residual, ys, XX, lik, Omega, pD] =  
nb_mlEstimator.varEstimator(y, X, options, init)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.bootstrapModel** ↑

```
[betaDraws, sigmaDraws, estOpt] = nb_nlsEstimator.bootstrapModel(model,...  
options, results, method, draws, iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.doTest** ↑

```
res = nb_nlsEstimator.doTest(res,options,residual)
```

Description:

Do different test based on the residuals.

See also:

[nb_nlsEstimator.estimate](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_nlsEstimator.estimate](#)** ↑

```
[results,options] = nb_nlsEstimator.estimate(options)
```

Description:

Estimate a model with nls.

Input:

- options : A struct on the format given by nb_nlsEstimator.template.
See also nb_nlsEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change.

See also:

[nb_nlsEstimator.print](#), [nb_nlsEstimator.help](#), [nb_nlsEstimator.template](#)

Written by Kenneth Sætherhagen Paulsen

► **[nb_nlsEstimator.getCoeff](#)** ↑

```
[coeff,numCoeff] = nb_nlsEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sætherhagen Paulsen

► **nb_nlsEstimator.getDependent** ↑

```
dependent = nb_nlsEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts or nb_cs object

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.getPredicted** ↑

```
predicted = nb_nlsEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts or nb_cs object

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.getResidual** ↑

```
residual = nb_nlsEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts or nb_cs object

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.help** ↑

```
helpText = nb_nlsEstimator.help
helpText = nb_nlsEstimator.help(option)
helpText = nb_nlsEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.makeArtificial ↑**

```
[YDRAW,start,finish,indY,startEst] = nb_nlsEstimator.makeArtificial(...  
    model,options,results,method,...  
    draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_nlsEstimator.print ↑**

```
res = nb_nlsEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_nlsEstimator.estimate function.
- options : A struct with the estimation options from the nb_nlsEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **`nb_nlsEstimator.template`** ↑

```
options = nb_olsEstimator.template()
```

Description:

Construct a struct which must be provided to the `nb_nlsEstimator.estimate` function.

This structure provided the user the possibility to set different estimation options.

Output:

- `options` : A struct.

Written by Kenneth Sætherhagen Paulsen

► **`nb_olsEstimator.addLags`** ↑

```
options = nb_olsEstimator.addLags(options)
```

Description:

Add lags to right hand side of estimation equation.

Written by Kenneth Sætherhagen Paulsen

► **`nb_olsEstimator.addLeads`** ↑

```
options = nb_olsEstimator.addLeads(options)
```

Description:

Add leads to right hand side of estimation equation. Only in the case that the unbalanced options is set to true.

Written by Kenneth Sætherhagen Paulsen

► **`nb_olsEstimator.addSeasonalDummies`** ↑

```
options = nb_olsEstimator.addSeasonalDummies(options)
```

Description:

Add seasonal dummies to model.

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.bootstrapModel ↑**

```
[betaDraws,sigmaDraws,estOpt] = nb_olsEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.doTest ↑**

```
res = nb_olsEstimator.doTest(res,options,beta,y,X,residual)
```

Description:

Do different test based on OLS residuals.

See also:

[nb_olsEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.estimate ↑**

```
[results,options] = nb_olsEstimator.estimate(options)
```

Description:

Estimate a model with ols.

Input:

- options : A struct on the format given by nb_olsEstimator.template.
See also nb_olsEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using the 'modelSelection' options.

See also:

[nb_olsEstimator.print](#), [nb_olsEstimator.help](#), [nb_olsEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_olsEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.getDependent** ↑

```
dependent = nb_olsEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.getPredicted** ↑

```
predicted = nb_olsEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.getResidual** ↑

```
residual = nb_olsEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.help** ↑

```
helpText = nb_olsEstimator.help  
helpText = nb_olsEstimator.help(option)  
helpText = nb_olsEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.makeArtificial** ↑

```
[YDRAW,start,finish,indY,startEst] = nb_olsEstimator.makeArtificial(...  
model,options,results,method,...  
draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.modelSelectionAlgorithm** ↑

```
options = nb_olsEstimator.modelSelectionAlgorithm(options,minLags)
```

Description:

Function to automatically select model.

See also:

[nb_olsEstimator.estimate](#), [nb_bVarEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.print** ↑

```
res = nb_olsEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the `nb_olsEstimator.estimate` function.
- options : A struct with the estimation options from the `nb_olsEstimator.estimate` function.
- precision : The precision of the printed result.

Output:

- res : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_olsEstimator.secureAllLags** ↑

```
options = nb_olsEstimator.secureAllLags(options)
```

Description:

Secure that all lags are added to the data, as they are needed for forecasting, shock decomposition etc. later.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_olsEstimator.template** ↑

```
options = nb_olsEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_olsEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_olsEstimator.varModifications** ↑

```
options = nb_olsEstimator.varModifications(options)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_pcaEstimator.estimate** ↑

```
[results,options] = nb_pcaEstimator.estimate(options)
```

Description:

Estimate a factors using principal components.

Input:

- options : A struct on the format given by nb_pcaEstimator.template.
See also nb_pcaEstimator.help.

Output:

```
- result : A struct with the estimation results.  
- options : The return model options, can change depending on inputs.
```

See also:

[nb_pcaEstimator.print](#), [nb_pcaEstimator.help](#), [nb_pcaEstimator.template](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_pcaEstimator.estimateFactors** ↑

No documentation

Written by Kenneth S. Paulsen

► **nb_pcaEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_pcaEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_pcaEstimator.getFactors** ↑

```
factors = nb_pcaEstimator.getFactors(results,options)
```

Description:

Get the estimated factors as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_pcaEstimator.getObservables** ↑

```
observables = nb_pcaEstimator.getObservables(results,options)
```

Description:

Get the observables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_pcaEstimator.help** ↑

```
helpText = nb_pcaEstimator.help  
helpText = nb_pcaEstimator.help(option)  
helpText = nb_pcaEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_pcaEstimator.print** ↑

```
res = nb_pcaEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_fmEstimator.estimate function.
- options : A struct with the estimation options from the nb_fmEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_pcaEstimator.template** ↑

```
options = nb_pcaEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_pcaEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.bootstrapModel** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_quantileEstimator.bootstrapModel(...  
model,options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.closeWaitbar** ↑

```
nb_quantileEstimator.closeWaitbar(h)
```

Description:

Close waitbar for producing posterior draws.

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.estimate** ↑

```
[results,options] = nb_quantileEstimator.estimate(options)
```

Description:

Estimate a model with quantile regression.

Input:

- options : A struct on the format given by
nb_quantileEstimator.template. See also
nb_quantileEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using
the 'modelSelection' options.

See also:

[nb_quantileEstimator.print](#), [nb_quantileEstimator.help](#)
[nb_quantileEstimator.template](#)

Written by Kenneth Sætherhagen Paulsen

► **nb_quantileEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_quantileEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sætherhagen Paulsen

► **nb_quantileEstimator.getDependent** ↑

```
dependent = nb_quantileEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.getPredicted** ↑

```
predicted = nb_quantileEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.getResidual** ↑

```
residual = nb_quantileEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.help** ↑

```
helpText = nb_quantileEstimator.help  
helpText = nb_quantileEstimator.help(option)  
helpText = nb_quantileEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_quantileEstimator.makeArtificial** ↑

```
[YDRAW,start,finish,indY,startEst] = nb_quantileEstimator.makeArtificial(...  
    model,options,results,method,...  
    draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_quantileEstimator.notifyWaitbar** ↑

```
nb_quantileEstimator.notifyWaitbar(h,ii,tt,note)
```

Description:

Notify waitbar for producing bootstrapped draws.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_quantileEstimator.notifyWaitbarQuantile** ↑

```
nb_quantileEstimator.notifyWaitbarQuantile(h,qq,note,numQ)
```

Description:

Notify waitbar for producing bootstrapped draws.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_quantileEstimator.openWaitbar** ↑

```
[h,note,isWaitbar] = nb_quantileEstimator.openWaitbar(waitbar,iter)
```

Description:

Open up waitbar for producing bootstrapped draws.

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.print** ↑

```
res = nb_quantileEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_quantileEstimator.estimate function.
- options : A struct with the estimation options from the nb_quantileEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.template** ↑

```
options = nb_olsEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_olsEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_quantileEstimator.varModifications** ↑

```
options = nb_olsEstimator.varModifications(options)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_realTimeEstimator.estimate** ↑

```
[results,options] = nb_realTimeEstimator.estimate(options)
```

Description:

Estimate a model with real-time data.

Caution : May run estimation in parallel on a given set of cores.

Input:

- options : A struct on the format given by underlying estimators template function. See for example nb_olsEstimator.template.

Output:

- result : A struct with the estimation results.

- options : The return model options, can change when for example using the 'modelSelection' options.

Written by Kenneth Sæterhagen Paulsen

► **nb_realTimeEstimator.mergeResults** ↑

```
res = nb_realTimeEstimator.mergeResults(res,resTemp)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_ridgeEstimator.bootstrapModel** ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_ridgeEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► nb_ridgeEstimator.estimate ↑

```
[results,options] = nb_ridgeEstimator.estimate(options)
```

Description:

Estimate a model with LASSO.

Input:

- options : A struct on the format given by nb_ridgeEstimator.template.
See also nb_ridgeEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example using
the 'modelSelection' options.

See also:

[nb_ridgeEstimator.print](#), [nb_ridgeEstimator.help](#)
[nb_ridgeEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► nb_ridgeEstimator.getCoeff ↑

```
[coeff,numCoeff] = nb_ridgeEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► nb_ridgeEstimator.getDependent ↑

```
dependent = nb_ridgeEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ridgeEstimator.getPredicted** ↑

```
predicted = nb_ridgeEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ridgeEstimator.getResidual** ↑

```
residual = nb_ridgeEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_ridgeEstimator.help** ↑

```
helpText = nb_ridgeEstimator.help  
helpText = nb_ridgeEstimator.help(option)  
helpText = nb_ridgeEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_ridgeEstimator.makeArtificial ↑**

```
[YDRAW,start,finish,indy,startEst] = nb_ridgeEstimator.makeArtificial(...  
    model,options,results,method,...  
    draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_ridgeEstimator.print ↑**

```
res = nb_ridgeEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_ridgeEstimator.estimate function.
- options : A struct with the estimation options from the nb_ridgeEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- res : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_ridgeEstimator.template ↑**

```
options = nb_ridgeEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_ridgeEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.bootstrapModel ↑**

```
[betaDraws,sigmaDraws,estOpt] = nb_rwEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.estimate ↑**

```
[results,options] = nb_rwEstimator.estimate(options)
```

Description:

Estimate a random walk model.

Input:

- options : A struct on the format given by nb_rwEstimator.template.
See also nb_rwEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options.

See also:

[nb_rwEstimator.print](#), [nb_rwEstimator.help](#), [nb_rwEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_rwEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.getDependent** ↑

```
dependent = nb_rwEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.getPredicted** ↑

```
predicted = nb_rwEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_rwEstimator.getResidual** ↑

```
residual = nb_rwEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_rwEstimator.help** ↑

```
helpText = nb_rwEstimator.help  
helpText = nb_rwEstimator.help(option)  
helpText = nb_rwEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_rwEstimator.makeArtificial** ↑

```
[YDRAW,start,finish,indY,startEst] = nb_rwEstimator.makeArtificial(...  
model,options,results,method,...  
draws,iter)
```

Description:

Make artificial data from model by simulation.

Caution: The options struct is assumed to already be index by iter!

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_rwEstimator.print** ↑

```
res = nb_rwEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_rwEstimator.estimate function.
- options : A struct with the estimation options from the nb_rwEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- res : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► nb_rwEstimator.template ↑

```
options = nb_rwEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_rwEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► nb_seasonalEstimator.estimate ↑

```
[results,options] = nb_seasonalEstimator.estimate(options)
```

Description:

Estimate seasonally adjusted series.

Input:

```
- options : A struct on the format given by  
          nb_seasonalEstimator.template.  
          See also nb_seasonalEstimator.help.
```

Output:

- result : A struct with the estimation results.
- options : The return model options, can change depending on inputs.

See also:

[nb_seasonalEstimator.print](#), [nb_seasonalEstimator.help](#)
[nb_seasonalEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_seasonalEstimator.estimateSeasonal ↑**

No documentation

Written by Kenneth S. Paulsen

► **nb_seasonalEstimator.getCoeff ↑**

```
[coeff,numCoeff] = nb_seasonalEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_seasonalEstimator.getDependent ↑**

```
dependent = nb_seasonalEstimator.getDependent(results,options)
```

Description:

Get the dependent variables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_seasonalEstimator.getSeasonallyAdjusted ↑**

```
seasonal = nb_seasonalEstimator.getSeasonallyAdjusted(results,options)
```

Description:

Get the estimated seasonally adjusted series as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_seasonalEstimator.help** ↑

```
helpText = nb_seasonalEstimator.help
helpText = nb_seasonalEstimator.help(option)
helpText = nb_seasonalEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_seasonalEstimator.print** ↑

```
res = nb_seasonalEstimator.print(results,options,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_seasonalEstimator.estimate function.
- options : A struct with the estimation options from the nb_seasonalEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_seasonalEstimator.template** ↑

```
options = nb_seasonalEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_seasonalEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_shorteningEstimator.doFunc** ↑

No documentation

Written by Kenneth S. Paulsen

► **nb_shorteningEstimator.doShortening** ↑

No documentation

Written by Kenneth S. Paulsen

► **nb_shorteningEstimator.estimate** ↑

```
[results,options] = nb_shorteningEstimator.estimate(options)
```

Description:

Shortening data according to settings.

Input:

- options : A struct on the format given by
nb_shorteningEstimator.template. See also
nb_shorteningEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change depending on inputs.

See also:

[nb_shorteningEstimator.print](#), [nb_shorteningEstimator.help](#)
[nb_shorteningEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_shorteningEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_shorteningEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_shorteningEstimator.getData** ↑

```
seasonal = nb_shorteningEstimator.getData(results,options)
```

Description:

Get the shortened original series as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_shorteningEstimator.getDependent** ↑

```
dependent = nb_shorteningEstimator.getDependent(results,options)
```

Description:

Get the dependent variables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_shorteningEstimator.help ↑**

```
helpText = nb_hpEstimator.help  
helpText = nb_hpEstimator.help(option)  
helpText = nb_hpEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **nb_shorteningEstimator.print ↑**

```
res = nb_hpEstimator.print(results,options,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_hpEstimator.estimate function.
- options : A struct with the estimation options from the nb_hpEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_shorteningEstimator.template** ↑

```
options = nb_shorteningEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_shorteningEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_statespaceEstimator.checkCalibration** ↑

```
estimated = nb_statespaceEstimator.checkCalibration(options,parser)
```

Description:

Check if all parameters has some values before we start the estimation.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_statespaceEstimator.estimate** ↑

```
[results,options] = nb_statespaceEstimator.estimate(options)
```

Description:

Estimate a state-space model using the NB Toolbox.

Input:

- options : A struct on the format given by
nb_statespaceEstimator.template. See also
nb_statespaceEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change when for example setting default estimation dates.

See also:

[nb_statespaceEstimator.print](#), [nb_statespaceEstimator.help](#)
[nb_statespaceEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.getBounds** ↑

```
[lb,ub] = nb_statespaceEstimator.getBounds(options)
```

Description:

Get lower and upper bound on priors.

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_statespaceEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.getDependent** ↑

```
dependent = nb_statespaceEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.getPredicted** ↑

```
predicted = nb_statespaceEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.getResidual** ↑

```
residual = nb_statespaceEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_statespaceEstimator.help** ↑

```
helpText = nb_statespaceEstimator.help  
helpText = nb_statespaceEstimator.help(option)  
helpText = nb_statespaceEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **`nb_statespaceEstimator.loadData`** ↑

```
[y,options] = nb_statespaceEstimator.loadData(options,parser)
```

Description:

Load data on observables.

Written by Kenneth Sæterhagen Paulsen

► **`nb_statespaceEstimator.print`** ↑

```
res = nb_statespaceEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the `nb_statespaceEstimator.estimate` function.
- options : A struct with the estimation options from the `nb_statespaceEstimator.estimate` function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **`nb_statespaceEstimator.sampler`** ↑

```
[betaD,sigmaD,posterior] = ...
nb_statespaceEstimator.sampler(posterior,draws)
```

Description:

Sample from the posterior of a estimated state space model.

See also:

[nb_drawFromPosterior](#), [nb_statespaceEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► nb_statespaceEstimator.template ↑

```
options = nb_statespaceEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_olsEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► nb_tslsEstimator.bootstrapModel ↑

```
[betaDraws,sigmaDraws,estOpt] = nb_tslsEstimator.bootstrapModel(model,...  
options,results,method,draws,iter)
```

Description:

Bootstrap model.

Written by Kenneth Sæterhagen Paulsen

► nb_tslsEstimator.estimate ↑

```
[results,options] = nb_tslsEstimator.estimate(options)
```

Description:

Estimate a model with two stage least squares.

Input:

- options : A struct on the format given by nb_tslsEstimator.template.
See also nb_tslsEstimator.help.

Output:

```
- result : A struct with the estimation results.  
- options : The return model options, can be updated.
```

See also:

[nb_tsEstimator.print](#), [nb_tsEstimator.help](#), [nb_tsEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_tsEstimator.getCoeff](#)** ↑

```
[coeff,numCoeff] = nb_tsEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **[nb_tsEstimator.getDependent](#)** ↑

```
residual = nb_tsEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **[nb_tsEstimator.getPredicted](#)** ↑

```
residual = nb_tsEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **[nb_tsEstimator.getResidual](#)** ↑

```
residual = nb_tsclsEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_tsclsEstimator.help ↑**

```
helpText = nb_tsclsEstimator.help  
helpText = nb_tsclsEstimator.help(option)  
helpText = nb_tsclsEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_tsclsEstimator.print ↑**

```
res = nb_tsclsEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_tsclsEstimator.estimate function.
- options : A struct with the estimation options from the nb_tsclsEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **nb_tslsEstimator.template** ↑

```
options = nb_olsEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_olsEstimator class constructor.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmpfsvEstimator.bootstrapModel** ↑

```
[betaDraws,factorDraws] =
    nb_tvpmpfsvEstimator.bootstrapModel(options,res,iter)
```

Description:

Not possible to bootstrap this model.

Written by Kenneth S. Paulsen

► **nb_tvpmpfsvEstimator.estimate** ↑

```
[results,options] = nb_tvpmpfsvEstimator.estimate(options)
```

Description:

Estimate a dynamic factor model with the algorithm of Schräder and Eraslan (2021), "Nowcasting GDP with a large factor model space".

Input:

- options : A struct on the format given by
nb_tvpmsvEstimator.template. See also
nb_tvpmsvEstimator.help.

Output:

- result : A struct with the estimation results.
- options : Some options may be updated.

See also:

[nb_tvpmsvEstimator.print](#), [nb_tvpmsvEstimator.help](#)
[nb_tvpmsvEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen and Maximilian Schröder

► **nb_tvpmsvEstimator.f_KFS_fac ↑**

[f_sm, H_t, ef_t, var_f_sm] = nb_tvpmsvEstimator.f_KFS_fac(prior, XY, ...
lambda_sm, beta_sm, Q_t_sm, V_t_sm)

Description:

This function is greatly inspired by Koop and Korobilis (2014) and adapted to the TVP-MF-DFM framework as described in the paper by Schroder and Eraslan (2021), "Nowcasting GDP with a large factor model space".

The purpose of this function is to estimate the factors conditional on the parameter estimates obtained from [nb_tvpmsvEstimator.f_KFS_params](#).

Input:

- priors : A struct, see the [nb_tvpmsvEstimator.getPriors](#) function.
- XY : The data on the observed variables.
- lambda_sm : The smoothed factor loadings estimates
- beta_sm : The smoothed VAR parameter estimates
- Q_t_sm : The smoothed Q_t estimates
- V_t_sm : The smoothed V_t estimates

Output:

- f_sm : The smoothed factor estimates
- H_t : The state-space matrix "H_t" containing all loadings (with aggregation scheme)
- ef_t : Measurement error.
- var_f_sm : Smoothed estimates of the one-step-ahead forecast error variance.

See also:

[nb_tvpmsvEstimator.normalEstimation](#)
[nb_tvpmsvEstimator.recursiveEstimation](#)

Written by Maximilian Schröder
Edited by Kenneth Sæterhagen Paulsen
- Formated the documentation to fit NB toolbox
- Changed inputs. Many of them are now calculated inside function instead.
- Returned the smoothed estimates of the one-step-ahead forecast error variance.

► **[nb_tvpmsvEstimator.f_KFS_params](#)** ↑

```
[lambda_sm,beta_sm,beta_t_sm,Q_t_sm,V_t_sm] =  
nb_tvpmsvEstimator.f_KFS_params(priors,f,XY)
```

Description:

This function is greatly inspired by Koop and Korobilis (2014) and adapted to the TVP-MF-DFM framework as described in the paper by Schroder and Eraslan (2021), "Nowcasting GDP with a large factor model space".

This function estimates the model parameters conditional on the preliminary factor estimates.

Input:

- priors : A struct, see the [nb_tvpmsvEstimator.getPriors](#) function.
- f : the preliminary factor estimates
- XY : the data on the observed variables.

Output:

- lambda_sm : the array of smoothed factor loadings estimates
- beta_sm : the reordered smoothed estimates of the VAR coefficients
- beta_t_sm : the matrix of smoothed estimate of the VAR coefficients
- Q_t_sm : the array of smoothed Q_t estimates
- V_t_sm : the array of smoothed V_t estimates

See also:

[nb_tvpmsvEstimator.normalEstimation](#)
[nb_tvpmsvEstimator.recursiveEstimation](#)

Written by Maximilian Schröder
Edited by Kenneth Sæterhagen Paulsen
- Formated the documentation to fit NB toolbox
- Changed inputs. Many of them are now calculated inside function instead.

► **nb_tpmfsvEstimator.f_em_mf ↑**

=====

DESCRIPTION

This program estimates a set of factors for a given dataset using principal component analysis. The number of factors estimated is determined by an information criterion specified by the user. Missing values in the original dataset are handled using an iterative expectation-maximization (EM) algorithm.

INPUTS

x = dataset (one series per column)
x_freq = frequency of each indicator
nFac = (predefined) number of factors
standardise = an integer indicating the type of transformation performed on each series in x before the factors are estimated.

OUTPUTS

ehat = difference between x and values of x predicted by the factors
Fhat = set of factors
lamhat = factor loadings
xhat = values of x predicted by the factors
eig = eigenvalues of $x3' * x3$ (where x3 is the dataset x post transformation and with missing values filled in)
y_em = x with missing values replaced from the EM algorithm

SUBFUNCTIONS

f_pca() - runs principal component analysis
f_standardise() - performs demeaning/standardisation of the data for PCA

BREAKDOWN OF THE FUNCTION

Part 1: Check that inputs are specified correctly.
Part 2: Setup.
Part 3: Initialize the EM algorithm -- fill in missing values with unconditional mean and estimate factors using the updated dataset.
Part 4: Perform the EM algorithm -- update missing values using factors, construct a new set of factors from the updated dataset, and repeat until the factor estimates do not change.

NOTES

Authors: Michael W. McCracken and Serena Ng
Date: 9/5/2017
Version: MATLAB 2014a
Required Toolboxes: None

Details for the three possible information criteria can be found in the paper "Determining the Number of Factors in Approximate Factor Models" by

Bai and Ng (2002).

The EM algorithm is essentially the one given in the paper "Macroeconomic Forecasting Using Diffusion Indexes" by Stock and Watson (2002). The algorithm is initialized by filling in missing values with the unconditional mean of the series, demeaning and standardizing the updated dataset, estimating factors from this demeaned and standardized dataset, and then using these factors to predict the dataset. The algorithm then proceeds as follows: update missing values using values predicted by the latest set of factors, demean and standardize the updated dataset, estimate a new set of factors using the demeaned and standardized updated dataset, and repeat the process until the factor estimates do not change.

Edited by Sercan Eraslan on 20/07/2020

The code is reduced to factor estimates with a fixed number of factors. The EM algorithm (part on updating missing values) is adjusted to incorporate mixed frequency data, such as monthly and quarterly according to Stock & Watson (2002) Appendix A.E p. 157.

=====

PART 1: CHECKS

► **nb_tvpmsvEstimator.f_factor_minnesotastruc ↑**

```
F_aux = nb_tvpmsvEstimator.f_factor_minnesotastruc(f_aux, lags_f, ...
n_f, t, k, p)
```

Description:

f_factor_minnesotastruc written by Maximilian Schröder. This function is mostly taken from Koop and Korobilis (2014) but modified to fit the current framework.

Input:

- f_aux : an auxilliary factor matrix of lagged factors
- lags_f : the number of factor lags
- n_f : the number of factors
- t : the time series length
- k : the size of the state vector
- p : the number of VAR parameters

Output:

- F_aux : the final auxilliary factor matrix

Written by Maximilian Schröder

Edited by Kenneth Sæterhagen Paulsen

- Formated the documentation to fit NB toolbox

► **nb_tvpmsvEstimator.f_smoothed_fac_shocks ↑**

```
[eps_f_sm] = nb_tvpmsvEstimator.f_smoothed_fac_shocks(priors, f_sm, beta_sm)
```

Description:

This function backs out the residuals of the factor state equation based on the smoothed factors and smoothed beta (factor VAR) coefficients.

Input:

- priors : A struct, see the nb_tvpmsvEstimator.getPriors function.
- f_sm : The smoothed factor estimates
- beta_am : the reordered smoothed estimates of the VAR coefficients (Companion form)

Output:

- eps_f_sm : A matrix containing the "smoothed" residuals of the factor state equation. The rows identify the factors, the columns identify the time dimension.

See also:

[nb_tvpmsvEstimator.print](#), [nb_tvpmsvEstimator.help](#)
[nb_tvpmsvEstimator.template](#)

Written by Maximilian Schräder

► **[nb_tvpmsvEstimator.getAggregationScheme](#)** ↑

```
agg = nb_tvpmsvEstimator.getAggregationScheme(options, index)
```

Description:

Get the mixed frequency aggregation scheme for the selected variable.

Inputs:

- options : A struct with estimation options. See [nb_tvpmsvEstimator.template](#) or [nb_tvpmsvEstimator.help](#).
- index : The index of the low frequency variable of interest.

Written by Kenneth S. Paulsen

► **[nb_tvpmsvEstimator.getBeta](#)** ↑

```
beta = nb_tvpmsvEstimator.getBeta(options, results)
```

Description:

Get all estimated coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmpfsvEstimator.getCoeff ↑**

```
[coeff,numCoeff] = nb_tvpmpfsvEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmpfsvEstimator.getDependent ↑**

```
dependent = nb_tvpmpfsvEstimator.getDependent(results,options)
```

Description:

Get the estimated model dependent values as a nb_ts object. There are none, see getObservables instead.

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmpfsvEstimator.getFactors ↑**

```
factors = nb_tvpmpfsvEstimator.getFactors(results,options)
```

Description:

Get the estimated factors of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmpfsvEstimator.getNumberOfCoeff ↑**

```
nCoeff = nb_tvpmpfsvEstimator.getNumberOfCoeff(options)
```

Description:

Get the number of estimated coefficients of the dynamic factor model.

Written by Kenneth S. Paulsen

► **nb_tvpmsvEstimator.getObservables** ↑

```
observables = nb_tvpmsvEstimator.getObservables(results,options)
```

Description:

Get the observables of the estimated model as a nb_ts object

Written by Kenneth Sætherhagen Paulsen

► **nb_tvpmsvEstimator.getPredicted** ↑

```
predicted = nb_tvpmsvEstimator.getPredicted(results,options)
```

Description:

Get the estimated model predicted values as a nb_ts object

Written by Kenneth Sætherhagen Paulsen

► **nb_tvpmsvEstimator.getPriors** ↑

```
priors = nb_tvpmsvEstimator.getPriors(options)
```

Description:

Get the priors and a few other settings needed for estimation.

Input:

- options : A struct on the format returned by
nb_tvpmsvEstimator.template.

Output:

- priors : A struct with fields;
 - > f_0 : the prior distribution of the factors (structure)
 - > lambda_0 : the prior distribution of the factor loadings
(structure)
 - > beta_0 : the prior distribution of the VAR coefficients

```
        (structure)
> V_0      : the prior of the obervation equation's variance
> Q_0      : the prior of the factor state equation's variance
> agg      : the aggregation scheme
> l_1m     : the decay factor for V_t^M
> l1_q     : the decay factor for V_t^Q
> l_2      : the decay factor for Q_t
> l_3      : the forgetting factor for W_t
> l_4      : the forgetting factor for R_t
> nLags    : the number of factor lags in the VAR
> n_m      : the number of indicators at monthly frequency
> n_q      : the number of indicators at quarterly frequency
```

See also:

[nb_tvpmsvEstimator.normalEstimation](#)
[nb_tvpmsvEstimator.recursiveEstimation](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmsvEstimator.getResidual** ↑

```
residual = nb_tvpmsvEstimator.getResidual(results,options)
```

Description:

Get the estimated model residuals as a nb_ts object. This is the idiosyncatic component of the measurement error, and not the residual of the transition equation.

Written by Kenneth Sæterhagen Paulsen

► **nb_tvpmsvEstimator.getStateNames** ↑

```
[stateNames,resNames] = nb_tvpmsvEstimator.getStateNames(options)
```

Description:

Constructs variable names for printing results.

Written by Kenneth S. Paulsen

► **nb_tvpmsvEstimator.help** ↑

```
helpText = nb_tvpmsvEstimator.help
helpText = nb_tvpmsvEstimator.help(option)
helpText = nb_tvpmsvEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sætherhagen Paulsen

► nb_tvpmpfsvEstimator.initialize ↑

```
results = nb_tvpmpfsvEstimator.initialize(options,X)
```

Description:

Initialization of the estimator of the dynamic factor model.

Inputs:

- options : A struct with estimation options. See
nb_tvpmpfsvEstimator.template or
nb_tvpmpfsvEstimator.help.
- X : A T x N double storing the data on the observables.

Written by Kenneth S. Paulsen

► nb_tvpmpfsvEstimator.normalEstimation ↑

```
[res,options] = nb_tvpmpfsvEstimator.normalEstimation(options,results)
```

Description:

Selects the wanted model class and estimate the model.

Inputs:

- options : A struct on the format returned by
nb_tvpmpfsvEstimator.template.

- results : A struct with initial conditions for EML iterations. Can be used for recursive estimation. If empty or not provided this struct is provided by nb_tvpmsvEstimator.initialize

Written by Kenneth S. Paulsen and Maximilian Schröder

► nb_tvpmsvEstimator.print ↑

```
res = nb_tvpmsvEstimator.print(results,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_tvpmsvEstimator.estimate function.
- options : A struct with the estimation options from the nb_tvpmsvEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Schröder Paulsen

► nb_tvpmsvEstimator.recursiveEstimation ↑

```
[results,options] = nb_tvpmsvEstimator.recursiveEstimation(options)
```

Description:

Selects the wanted model class and recursively estimate the model.

Written by Kenneth S. Paulsen and Maximilian Schröder

► nb_tvpmsvEstimator.template ↑

```
options = nb_tvpmsvEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_tvpmsvEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **nb_whitenEstimator.estimate** ↑

```
[results,options] = nb_whitenEstimator.estimate(options)
```

Description:

Estimate a factors that have variance one and covariances 0 using whitening.

Input:

- options : A struct on the format given by nb_whitenEstimator.template.
See also nb_whitenEstimator.help.

Output:

- result : A struct with the estimation results.
- options : The return model options, can change depending on inputs.

See also:

[nb_whitenEstimator.print](#), [nb_whitenEstimator.help](#)
[nb_whitenEstimator.template](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_whitenEstimator.estimateFactors** ↑

No documentation

Written by Kenneth S. Paulsen

► **nb_whitenEstimator.getCoeff** ↑

```
[coeff,numCoeff] = nb_whitenEstimator.getCoeff(options)
```

Description:

Get names of coefficients.

Written by Kenneth Sæterhagen Paulsen

► **nb_whitenEstimator.getFactors ↑**

```
factors = nb_whitenEstimator.getFactors(results,options)
```

Description:

Get the estimated factors as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_whitenEstimator.getObservables ↑**

```
observables = nb_whitenEstimator.getObservables(results,options)
```

Description:

Get the observables of the estimated model as a nb_ts object

Written by Kenneth Sæterhagen Paulsen

► **nb_whitenEstimator.help ↑**

```
helpText = nb_whitenEstimator.help  
helpText = nb_whitenEstimator.help(option)  
helpText = nb_whitenEstimator.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_whitenEstimator.print** ↑

```
res = nb_whitenEstimator.print(results,options,precision)
```

Description:

Get the estimation results as a char.

Input:

- results : A struct with the estimation results from the nb_fmEstimator.estimate function.
- options : A struct with the estimation options from the nb_fmEstimator.estimate function.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_whitenEstimator.template** ↑

```
options = nb_whitenEstimator.template()
```

Description:

Construct a struct which must be provided to the nb_whitenEstimator.estimate function.

This structure provided the user the possibility to set different estimation options.

Output:

- options : A struct.

Written by Kenneth SÃ¶terhagen Paulsen

◆ nb_loadDraws

```
posterior = nb_loadDraws(pathToLoad)
```

Description:

Load posterior draws from file.

Written by Kenneth Sæterhagen Paulsen

◆ nb_saveDraws

```
pathToSave = nb_saveDraws(modelName,posterior)
```

Description:

Save posterior draws to file, and return the saved location.

Written by Kenneth Sæterhagen Paulsen

◆ nb_addStars

```
tests = nb_addStars(results,tests,indNT,indW,indARCH,indAT)
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_almonLag

```
x = nb_almonLag(theta,K)
```

Description:

Implementation of the Almon lag polynominal.

Input:

- theta : A 1 x Q x P double with the hyperparameters of the almon lag polynominal; $\exp(\theta_1 \cdot k + \dots + \theta_Q \cdot k^Q) / \sum_k \exp(\theta_1 \cdot k + \dots + \theta_Q \cdot k^Q)$
- K : Number of lags of the polynominal.

Output:

- x : A 1 x K double with the value of the Almon lag polynominal for $k = 1:K$.

See also:

[nb_midasFunc](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_almonPoly

`Q = nb_almonPoly(P, K, E)`

Description:

Implementation of the Almon lag polynominal.

Input:

- `P` : Number of lags of the polynomial. If empty it will default to `K`. Either as a scalar integer or a vector of integers with length `E`.
- `K` : Number of lags of the regressor. Either as a scalar integer or a vector of integers with length `E`.
- `E` : Number of variables.

Output:

- `Q` : A $P \times E$ x $K \times E$ (or $\text{sum}(P(i)) \times \text{sum}(K(i))$) double with the mapping matrix.

See also:

[nb_midasFunc](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_arimaFunc

`results = nb_arimaFunc(y)`
`results = nb_arimaFunc(y, p, i, q, sp, sq, varargin)`

Description:

Estimate a ARIMA(p, i, q) model to a time-series represented as a double.

$$\begin{aligned}y(t) &= b * \text{exo} + u(t) \\u(t) &= \lambda * u(t-1) + c * \text{exo} + \epsilon(t)\end{aligned}$$

If the `p`, `i` or `q` are given by nan values they are being selected by this function.

i is found by the adf unit root test. I.e. the first degree of integration which reject the null of a unit root.

p and q are found by the minimizing the model selection criterion provided by the 'criterion' input. Default is to use the corrected Akaike information criterion. ('aicc')

Input:

- y : A nobs x 1 double vector with the time-series.
- p : The AR degree. As a number. Provide nan if you want to use a model selection criterion to find it.
- i : The degree of integration. I.e. the number of times the time-series have to be differenced. Provide nan if you want to use a unit root test to find it.
Will use the Augmented Dickey-Fuller test for unit root.
- q : The MA degree. As a number. Provide nan if you want to use a model selection criterion to find it.
- sp : Seasonal autoregressive (SAR) component. E.g. 0 to exclude, 4 for quarterly data, 12 for monthly. Only an option if 'method' is set to 'ml'.
- sq : Seasonal moving average (SMA) component. E.g. 0 to exclude, 4 for quarterly data, 12 for monthly. Only an option if 'method' is set to 'ml'.

Optional inputs:

- 'constant' : Give 1 to include constant, otherwise give 0.
Default is 0.
- 'criterion' : The criterion to use when selecting between ARIMA models. See the function nb_infoCriterion for more. Default is 'aicc'.
- 'exo' : Exogenous regressors in the observation equation. As a nobs x nvars. Defualt is [].
- 'texo' : Exogenous regressors in the transition equation. As a nobs x mvars. Defualt is [].
- 'maxAR' : Maximal number of AR coefficient. Default is 3.
- 'maxMA' : Maximal number of MA coefficient. Default is 3.
- 'method' : Either:
 - > 'ml' : Maximum likelihood
 - > 'hr' : Hannan-Rissanen algorithm. Default.
(Same as OLS if only AR terms are included)
- 'optimizer' : See the output from the nb_getOptimizers('arima').

- 'filter' : Give 1 if you want to run the kalman filter to estimate the residual. Otherwise give 0. Default is not to run filter. Only for 'ml'.
- 'alpha' : The significance level used by the ADF test for unit root when i is set to empty. Default is 0.05.
- 'test' : Test for roots outside the unit circle. Default is true. Must be true or false.
- 'options' : Inputs given to the optimset function. If no optional inputs are given default settings are used. Must be given as a struct or a cell array. If given as a cell array it is given to the optimset function. struct is recommended. See nb_getDefaultOptimset.
- 'covrepair' : See help on nb_mlarima. Only an option if 'method' is set to 'ml'.

Output:

- results : A struct consisting of:
 - > beta : The estimated coefficients. Order; constant, AR, MA, (SAR, SMA then exogenous).
 - > stdBeta : The standard deviation of the estimated coefficients. Constant, AR coefficients then MA coefficients.
 - > tStatBeta : The t-statistics of the estimated coefficients. Constant, AR coefficients then MA coefficients.
 - > pValBeta : The p-values of the estimated coefficients. Constant, AR coefficients then MA coefficients.
 - > likelihood : The log likelihood.
 - > i : The degree of integration.
 - > residual : The estimated residual. As a nsample x 1 double.
 - > X : The model regressors. As a nsample x nlags double. Is empty if 'method' is set to 'ml'
 - > y : The model dependent variable. As a nsample x 1 double.
 - > u : See same output of nb_mlarima or nb_hrarima.
 - > z : See same output of nb_mlarima or nb_hrarima.
 - > AR : Number of AR terms.

> MA : Number of MA terms.

See also:

[nb_getDefaultOptimset](#), [nb_getOptimizers](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_arimaStateSpace**

```
[x0,P0,d,H,R,T,c,A,B,Q,G,obs,failed] = nb_arimaStateSpace(par,p,q,sp,...  
sq,constant,nExoT,stabilityTest)
```

Description:

Returns a state-space representation of an ARIMA(p,0,q,sq,sq) model with seasonal autoregressive and seasonal moving average terms.

Observation equation:

$$y = d + Hx + Tz + v$$

State equation:

$$x = c + Ax_{-1} + Gz + Bu$$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q

$v \sim N(0, R)$ meaning v is gaussian noise with covariance R

Input:

- par : The parameter vector of the ARIMA model. Order:
 - constant
 - AR terms
 - MA terms
 - SAR terms
 - SMA terms
 - Exogenous regressors
 - Std of residual
- p : The number of AR terms.
- q : the number of MA terms.
- constant : Include constant or not. true or false.
- stabilityTest : Check stability of system. Sets failed to true if model is not stationary. Default is false.
- nExoT : Number of exogenous variables included in the state equation.

Output:

- `x0` : Initial state vector.
- `P0` : Initial variance.
- See equation above
- `obs` : Index of observables in the state vector.
- `failed` : See stabilityTest input. true if model fail test.

See also:

[nb_kalmanlikelihood](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_bayesEst**

```
[exitflag,beta,fval,H] = nb_bayesEst(init,ub,lb,opt,optimizer,...  
func,priorFunc,varargin)
```

Description:

Estimate multiple (related) equations using Bayesian (mode) estimation.

The minimized objective is: `func(beta) - priorFunc(beta)`.

Input:

- `init` : Initial values of the parameters of the model to estimate.
As a `nParam x 1 double`.
- `ub` : Upper bound on the parameters of the model to estimate.
As a `nParam x 1` or `nParam x 1 double`. If given as empty
`inf` is default. Only applies if optimizer is set to
`'fmincon'` or `'nb_abc'`.
- `lb` : Lower bound on the parameters of the model to estimate.
As a `nParam x 1` or `nParam x 1 double`. If given as empty
`-inf` is default. Only applies if optimizer is set to
`'fmincon'` or `'nb_abc'`.
- `opt` : See the optimset function for more on this input. Default
values will be used if it is given as `[]`.
- `optimizer` : The optimizer to be used; `'fminunc'`, `'fminsearch'`,
`'fmincon'` or `'nb_abc'`. `'fmincon'` is default.
- `func` : A function handle that takes the inputs in the following
order:
> The parameter matrix as a `(nParam x nEq) x 1 double`,
i.e. `vec(beta)`.
> The `y` input.
> The `x` input.
> The rest of the optional inputs given to this function

(varargin)

The output should be a 1 x 1 double with minus the log likelihood of the model.

- priorFunc : A function handle that takes the inputs in the following order:
 - > The parameter matrix as a (nParam x nEq) x 1 double, i.e. vec(beta).

The output should be a 1 x 1 double log prior evaluated at the current parameter values.

Optional input:

- 'NONLCON' : A function handle that return the values of C(beta) and Ceq(beta), where C(beta) <= 0 and Ceq(beta) == 0. E.g. [C,Ceq] = nonlinconstr(beta).
For more see; help nb_abc.constraints
- 'highValue' : The highest possible value for the objective being minimized. Default is 1000000.
- varargin : The rest will be given as optional inputs to the func input.

Output:

- exitflag : The reason for exiting the minimization. Positive numbers and 0 means success, otherwise failure.
- beta : A nxvar x neq matrix with the estimated parameters.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- H : Hessian matrix at the mode of the objective function.
As a

See also
nb_midasFunc

Written by Kenneth S. Paulsen

◆ **nb_betaLag**

x = nb_betaLag(theta, K)

Description:

Implementation of the Beta lag polynominal.

Input:

- theta : A 1 x Q x P double with the hyperparameters of the Beta lag polynominal. See page 5 of Ghysels et al. (2006). In equation (3) we have used K+1 instead of K. Q must be equal to 2.
- K : Number of lags of the polynominal.

Output:

- x : A 1 x K double with the value of the Beta lag polynominal for k = 1:K.

See also:

[nb_midasFunc](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_betaPoly**

`Q = nb_betaPoly(theta,K,E)`

Description:

Implementation of the Beta lag polynominal.

Input:

- theta : A scalar double with the last hyperparameter of the Beta lag polynominal. See page 5 of Ghysels et al. (2006). In equation (3) we have used K+1 instead of K. The first hyperparameter is assumed to be 1!
- P : Number of lags of the polynomial. If empty it will default to K. Either as a scalar integer or a vector of integers with length E.
- K : Number of lags of the regressor. Either as a scalar integer or a vector of integers with length E.
- E : Number of variables.

Output:

- Q : A E x K*E (or E x sum(K(i))) double with the mapping matrix.

See also:

[nb_betaProfiling](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_betaProfiling

```
[b,e,X_tilda] = nb_betaProfiling(y,X,constant,AR,nExo,nLags,upper,nGrid)
```

Description:

Beta profiling as in Ghysels and Qian (2019), Estimating MIDAS regressions via OLS with polynomial parameter profiling.

Input:

- y : Left hand side variable of the model. At low frequency.
With size T x 1.
- X : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression. These are of a higher frequency than y. Lags of regressors should already be added. The nExo input must be set.

The order of the regressors must be;
var1_lag1, var2_lag1, var1_lag2, var2_lag2, var1_lag3
i.e. they can have different number of lags. See the nLags input!
- constant : Set to true to add constant term to the regression model.
Caution: Is not assumed to be included in X!
- AR : Set to true to indicate that the first column of X is the lagged series of y.
- nExo : Number of exogenous variables excluding the lags.
- nLags : Either a scalar integer with the number of lags of all the regressors, or a vector of integers with length nExo with the number of lags of each regressor.
- upper : The upper limit on the grid of the second parameter of the beta lag polynomial. Default is 40. Lower limit is always set to 1!
- nGrid : The number of grid points profiled over. Default is 80.

Output:

- b : The estimated parameters of the model $y = X_{\text{tilda}} \cdot \beta + e$. The ordering is constant term, AR term, the OLS estimates and the last element is the selected value of the profiled parameter beta lag polynomial. Size is constant + AR + nExo x 1.
- e : The residual from the regression model $y = X_{\text{tilda}} \cdot \beta + e$. With size T x 1.
- X_tilda : The regressors of the model $y = X_{\text{tilda}} \cdot \beta + e$. With size T x (constant + AR + nExo).

Written by Kenneth Sætherhagen Paulsen

◆ nb_checkSample

```
nb_checkSample(y,X)
[endInd,startInd,varargout] = nb_checkSample(varargin)
```

Description:

Check data first and last real observations.

Input:

- varargin : Matrices that can be horcat

Output:

- endInd : Last real data point in the data
- startInd : First real data point in the data
- varargout : Same as varargin, but may have been shortened due to trailing and leading nan values

Written by Kenneth Sætherhagen Paulsen

◆ nb_drawFromPosterior

```
[beta,sigma]      = nb_drawFromPosterior(posterior,draws,waitbar)
[beta,sigma,yD,pD] = nb_drawFromPosterior(posterior,draws,waitbar)
```

Description:

Make draws from a posterior.

Caution: Will use the last parameter draw given by the `posterior.betaD(:,:,end)` and `posterior.sigmaD(:,:,end)`, where that applies.

Input:

- posterior : A struct with the needed inputs to the different functions to make posterior draws. See the posterior output of;
 - > `nb_bVarEstimator.minnesota`
 - > `nb_bVarEstimator.minnesotaMF`
 - > `nb_bVarEstimator.jeffrey`
 - > `nb_bVarEstimator.laplace`
 - > `nb_bVarEstimator.glp`
 - > `nb_bVarEstimator.glpMF`

```

> nb_bVarEstimator.nwishart
> nb_bVarEstimator.nwishartMF
> nb_bVarEstimator.inwishart
> nb_bVarEstimator.inwishartMF
> nb_bVarEstimator.laplace
> nb_bVarEstimator.horseshoe
> nb_bVarEstimator.dsge
> nb_risedsgeEstimator.sampler
> nb_statespaceEstimator.sampler
> nb_dsge.priorPredictiveAnalysis

- nDraws      : Number of draws from the posterior distribution

- waitbar    : true, false or an object of class nb_waitbar5.

```

Output:

```

- beta       : The posterior draws of the model coefficients, as a nCoeff
               x nEq x nDraws double.

- sigma      : The posterior draws of the residual covariance matrix as a
               nEq x nEq x nDraws double.

- yD         : Sampling from the posterior of missing observations. [] if
               models does not handle missing observations.

- pD         : The covariance matrix of the missing observations at the
               end of the sample. As a nEq x nEq x nNowcast x nDraws
               double, where nNowcast is the number of missing
               observations from where all endogenous variables has
               non-missing observations.

```

See also:

[nb_bVarEstimator](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_estvarols**

```
varargout = nb_estvarols(y,nlag,x,constant)
```

Description:

Estimate a VAR model using OLS

Input:

```

- y          : A nobs x nvar matrix with endogenous regressors.

- nlag       : The number of lags of the VAR. Default is 1.

- constant   : If a constant should be included in the regression.
               Default is 1, i.e. include constant.

```

- timeTrend : If a linear time trend is wanted in the estimation.
Will be added first/second in the right hand side
variables. Default is not to add it (0).
- x : A nobs x nxvar matrix with exogenous regressors.
- stdType : - 'w' : Will return White heteroskedasticity
robust standard errors.
- 'nw' : Will return Newey-West heteroskedasticity
and autocorrelation robust standard errors.

To estimate the covariance matrix of the estimated parameters the bartlett kernel is used with the bandwidth set to $\text{floor}(4(T/100)^{(2/9)})$ as Eviews also uses.

- 'h' : Will return homoskedasticity only robust standard errors. Default.

Output:

- beta : A nxvar x neqs vector with the estimated parameters.
- stdBeta : A nxvar x neqs vector with the standard deviation of the estimated parameters.
- tStatBeta : A nxvar x neqs vector with t-statistics of the estimated parameters.
- pValBeta : A nxvar x neqs vector with the p-values of the estimated parameters.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- x : Regressors including constant and time-trend.

Written by Kenneth S. Paulsen

◆ nb_getMidasIndex

```
ind = nb_getMidasIndex(I,K,E)
```

Description:

Get index of the lags of regressor I in the list of all regressors in a MIDAS model.

Input:

- I : The variable number. In [1,E].
- K : Number of lags of the regressor. As a vector of integers with

length E.

- E : The number of regressors in the MIDAS model.

Output:

- ind : The index of the regressors associated with the variable I.

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_homoOnlySTD**

```
stdBeta = nb_homoOnlySTD(x,residual)
```

Description:

Estimation of homoscedasticity only standard errors.

Input:

- x : The regressors of the equation, as a nobs x nvar double.
- u : Residual from regression, as a nobs x neq double.
- xpxi : $(x'x)^{-1}$. If not given it will be calculated.
- restr : A 1 x neq where each element is a 1 x nxvar + extra logical. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included. Each logical element must be true to include in the regression equation of the corresponding equation.

Output:

- stdBeta : Std as a nvar x neq double.

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_hrarima**

```
results = nb_hrarima(y,p,i,q,constant)
results = nb_hrarima(y,p,i,q,constant,test,z,x)
```

Description:

Estimate a ARIMA(p,i,q) or ARIMAX(p,i,q) model using the Hannan-Rissanen algorithm.

$$\begin{aligned}y(t) &= b \cdot z + u(t) \\u(t) &= \lambda \cdot u(t-1) + c \cdot x + \epsilon(t)\end{aligned}$$

Input:

```
- y          : A nobs x 1 double vector with the time-series.  
- p          : The AR degree. As a number.  
- i          : The degree of integration. I.e. the number of times  
               the time-series have to be differenced.  
- q          : The MA degree. As a number.  
- constant   : Give 1 if a constant should be included, otherwise  
               0. Default is 1.  
- test       : Give error if any of the roots of the AR process is outside  
               the unit circle.  
- z          : Exogenous regressors in the observation equation. As a  
               nobs x nvars. Default is [].  
- x          : Exogenous regressors in the transition equation. As  
               a nobs x mvars. Default is [].
```

Output:

```
- results : A struct consisting of:  
  
> beta      : The estimated coefficient of the  
               constant. Constant, AR coefficients  
               then MA coefficients.  
  
> stdBeta   : The standard deviation of the estimated  
               coefficients. Constant, AR  
               coefficients then MA coefficients.  
  
> tStatBeta : The t-statistics of the estimated  
               coefficients. Constant, AR  
               coefficients then MA coefficients.  
  
> pValBeta  : The p-values of the estimated  
               coefficients. Constant, AR  
               coefficients then MA coefficients.  
  
> sigma      : Estimated std of the residual.  
  
> likelihood : The log likelihood.  
  
> residual   : The estimated residual. As a  
               nsample x 1 double.  
  
> X          : The model regressors. As a  
               nsample x nlags double.  
  
> y          : The model dependent variable. As a  
               nsample x 1 double.  
  
> u          : The residual from the first stage regression  
               of the dependent variable on the exogenous  
               and constant. If no constant and z is empty
```

this will be equal to y.

> z : The exogenous variables included in the observation equation.

> x : The exogenous variables included in the transition equation.

Written by Kenneth Sæterhagen Paulsen

◆ nb_jeffrey

```
beta          = nb_jeffrey(y,x)
[beta,sigma,residual,X] = nb_jeffrey(y,x,prior,constant,timeTrend, ...
    restrictions)
```

Description:

Estimate multiple (related) equations using Jeffrey priors.

$y = X\beta + \text{residual}$, $\text{residual} \sim N(0, \sigma)$ (1)

Prior on beta : Diffuse

Prior on sigma : Diffuse

Input:

- y : A double matrix of size T x Q of the dependent variable of the regression(s).

- x : A double matrix of size T x N of the right hand side variables of all equations of the regression.

- prior : A structure with the prior specification. If not given the default priors are:
> draws : 1

For more on this prior see nb_jeffreyPrior.

- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false.

- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second). Default is false.

- restrictions :

Output:

- beta : A (extra + nxvar) x nEq x draws matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.

- sigma : A nEq x nEq x draws matrix with the covariance matrix of residuals.
- residual : Residual from the estimated equation. As an nobs x nEq matrix. If draws > 1, this is calculated at the mean.
- X : Regressors including constant and time-trend. If more than one equation this will be given as kron(I,x).

See also
[nb_jeffreyPrior](#), [nb_bayesEstimator](#), [nb_singleEq](#)

Written by Kenneth S. Paulsen

◆ **nb_jeffreyPrior**

prior = nb_jeffreyPrior(varargin)

Description:

Set the priors of the type:

y = X*beta + residual, residual ~ N(0,sigma) (1)

y has size T x Q
X has size T x N

Prior on beta : Diffuse
Prior on sigma : Diffuse

Optionsl input:

- Run without inputs to get default priors.
- Run with 'list' as the first input to get a cellstr with the supported prior options.
- Run with 'help' to get a char with help on each option.
- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:

[nb_jeffrey](#)

◆ nb_kalmanLikelihoodBreakPointDSGE

```
lik = nb_kalmanLikelihoodBreakPointDSGE(par,model,y,varargin)
```

Description:

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. A cell with size 1 x nStates. Each element as a nEndo x nEndo double.
 - > C : The impact of shock matrix. A cell with size 1 x nStates. Each element as a nEndo x nExo double.
 - > x : Initial state vector. As a nEndo x 1 double.
 - > P : Initial covariance of state variables, as a nEndo x nEndo double.
 - > options : A struct with the fields:
 - * kf_riccatiTol : Converge criteria on the Kalman gain.
 - * kf_preset : Discarded observation of the likelihood at the begining.
 - * states : A nobs vector with the states at the given period.
 - * ss : The steady-state of the model in each regime. As a cell array.
 - > err : A empty string if success, otherwise the error message as a char (lik = 1e10 in this case).

- y : Observation vector. A nObs x T double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodDSGE](#), [nb_kalmanSmoothenBreakPointDSGE](#)

Written by Kenneth Åström Paulsen

◆ **nb_kalmanLikelihoodDSGE**

lik = nb_kalmanLikelihoodDSGE(par, model, y, varargin)

Description:

The model function must return the given matrices in the system below:

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A*x(t-1) + C*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.

- model : A function handle returning the following matrices (in this order):

> H : Observation matrix. As a nObs x nEndo double.
> A : State transition matrix. As a nEndo x nEndo double.
> C : The impact of shock matrix. As a nEndo x nExo double.
> x : Initial state vector. As a nEndo x 1 double.

```

> P      : Initial covariance of state variables, as a nEndo x
           nEndo double.

> options : A struct with the fields:

    * kf_riccatiTol : Converge criteria on the Kalman
           gain.

    * kf_presample  : Discarded observation of the
           likelihood at the begining.

> err     : A empty string if success, otherwise the error
           message as a char (lik = le10 in this case).

- y : Observation vector. A nObs x T double.

```

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodMissingDSGE](#), [nb_kalmanSmoothenDSGE](#)

Written by Kenneth Å!terhagen Paulsen

◆ **nb_kalmanLikelihoodDiffuseBreakPointDSGE**

lik = nb_kalmanLikelihoodDiffuseBreakPointDSGE(par,model,y,varargin)

Description:

Diffuse piecewise linear Kalman filter. With potentially missing observations!

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = s(t) + A(t) \cdot (x(t-1) - s(t)) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

```

- par    : A parameter vector which is given as the first input to
          the function handle given by the model input.

- model : A function handle returning the following matrices (in
          this order):

    > H      : Observation matrix. As a nObs x nEndo double.

    > A      : State transition matrix. A cell with size 1 x
              nStates. Each element as a nEndo x nEndo double.

    > C      : The impact of shock matrix. A cell with size 1 x
              nStates. Each element as a nEndo x nExo double.

    > x      : Initial state vector. As a nEndo x 1 double.

    > P      : Initial state variance. As a nStationaryVar x
              nStationaryVar double.

    > Pinf   : Initial state variance of the non-stationary
              variables. As a nNonStationaryVar x
              nNonStationaryVar double.

    > options : A struct with the fields:

        * kf_riccatiTol : Converge criteria on the Kalman
                           gain.

        * kf_presample  : Discarded observation of the
                           likelihood at the begining.

        * states         : A nobs vector with the states
                           at the given period.

        * ss             : The steady-state of the model in
                           each regime. As a cell array.

    > err    : A empty string if success, otherwise the error
              message as a char (lik = 1e10 in this case).

- y : Observation vector. A nObs x T double.

```

Optional input:

```
- varargin : Optional inputs given to the function handle given
            by the model input.
```

Output:

```
- lik : Minus the log likelihood. As a 1 x 1 double.
```

See also:

[nb_kalmanLikelihoodBreakPointDSGE](#), [nb_setUpForDiffuseFilter](#)
[nb_kalmanSmootherDiffuseBreakPointDSGE](#)
[nb_kalmanLikelihoodUnivariateBreakPointDSGE](#)

◆ nb_kalmanLikelihoodDiffuseDSGE

```
lik = nb_kalmanLikelihoodDiffuseDSGE(par,model,y,varargin)
```

Description:

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + B \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance Q

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. As a nEndo x nEndo double.
 - > C : The impact of shock matrix. As a nEndo x nExo double.
 - > x : Initial state vector. As a nEndo x 1 double.
 - > P : Initial state variance. As a nStationaryVar x nStationaryVar double.
 - > Pinf : Initial state variance of the non-stationary variables. As a nNonStationaryVar x nNonStationaryVar double.
 - > options : A struct with the fields:
 - * kf_riccatiTol : Convergence criteria on the Kalman gain.
 - * kf_presample : Discarded observation of the likelihood at the beginning.
- y : Observation vector. A nObs x T double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodDSGE](#), [nb_kalmanSmoothenDiffuseDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanLikelihoodUnivariateDSGE](#)

Written by Kenneth Åkerhagen Paulsen

◆ **nb_kalmanLikelihoodDiffuseStochasticTrendDSGE**

lik = nb_kalmanLikelihoodDiffuseStochasticTrendDSGE(par,estStruct)

Description:

Diffuse piecewise linear Kalman filter with updating of the "steady state" based on some stochastic trends of the observation part of the model. Will also handle potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + B(t) \cdot z(t) + C(t) \cdot u(t)$$

where $ss(t) = G(theta, x(t-1))$ and $A(t)$, $B(t)$ and $C(t)$ are the solution around this updated approximation point (or steady state).

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : Current vector of the estimated parameters.
- estStruct : See the `nb_dsge.getObjectiveForEstimation` method.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

- A : The updated estimates of the transition matrix.
- B : The updated estimates of the contant term of the state equation.
- C : The updated estimates of the shock impact matrix.
- ss : The updated estimates of the approximation point (steady state).
- p : The updated estimates of the parameters of the model.

See also:

[nb_kalmanSmotherDiffuseStochasticTrendDSGE](#), [nb_setUpForDiffuseFilter](#)
[nb_kalmanLikelihoodUnivariateStochasticTrendDSGE](#), [nb_dsge.updateSolution](#)

Written by Kenneth Sæterhagen Paulsen.

◆ **nb_kalmanLikelihoodDiffuseTVPDSGE**

lik = nb_kalmanLikelihoodDiffuseTVPDSGE(par,model,y,varargin)

Description:

Diffuse Kalman filter with observed time-varying parameters that does not affect the steady state.

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A(t) \cdot x(t-1) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. As a nEndo x nEndo x T double.
 - > C : The impact of shock matrix. As a nEndo x nExo x T double.
 - > x : Initial state vector. As a nEndo x 1 double.

```

> P      : Initial state variance. As a nStationaryVar x
           nStationaryVar double.

> Pinf   : Initial state variance of the non-stationary
           variables. As a nNonStationaryVar x
           nNonStationaryVar double.

> options : A struct with the fields:

          * kf_riccatiTol : Converge criteria on the Kalman
                             gain.

          * kf_presample  : Discarded observation of the
                             likelihood at the begining.

> err     : A empty string if success, otherwise the error
           message as a char (lik = 1e10 in this case).

- y : Observation vector. A nObs x T double.

```

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodTVPDSGE](#), [nb_kalmanSmoothenDiffuseTVPDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanLikelihoodUnivariateTVPDSGE](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_kalmanLikelihoodMissingDSGE**

`lik = nb_kalmanLikelihoodMissingDSGE(par, model, y, varargin)`

Description:

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + C \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering ans smoothing for multivariate state space models".

Input:

```
- par    : A parameter vector which is given as the first input to  
          the function handle given by the model input.  
  
- model : A function handle returning the following matrices (in  
          this order):  
  
    > H      : Observation matrix. As a nObs x nEndo double.  
    > A      : State transition matrix. As a nEndo x nEndo double.  
    > C      : The impact of shock matrix. As a nEndo x nExo  
          double.  
    > x      : Initial state vector. As a nEndo x 1 double.  
    > P      : Initial covariance of state variables, as a nEndo x  
          nEndo double.  
  
    > options : A struct with the fields:  
  
        * kf_riccatiTol : Converge criteria on the Kalman  
                          gain.  
  
        * kf_presample  : Discarded observation of the  
                          likelihood at the begining.  
  
    > err     : A empty string if success, otherwise the error  
          message as a char (lik = 1e10 in this case).  
  
- y : Observation vector. A nObs x T double.
```

Optional input:

```
- varargin : Optional inputs given to the function handle given  
          by the model input.
```

Output:

```
- lik : Minus the log likelihood. As a 1 x 1 double.
```

See also:

[nb_kalmanLikelihoodDSGE](#), [nb_kalmanSmoothenMissingDSGE](#)

Written by Kenneth Åkerhagen Paulsen

◆ nb_kalmanLikelihoodTVPDSGE

```
lik = nb_kalmanLikelihoodTVPDSGE(par,model,y,varargin)
```

Description:

Kalman filter with observed time-varying parameters that does not affect the steady state.

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A(t) \cdot x(t) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. As a nEndo x nEndo x T double.
 - > C : The impact of shock matrix. As a nEndo x nExo x T double.
 - > x : Initial state vector. As a nEndo x 1 double.
 - > P : Initial covariance of state variables, as a nEndo x nEndo double.
 - > options : A struct with the fields:
 - * kf_riccatiTol : Converge criteria on the Kalman gain.
 - * kf_presample : Discarded observation of the likelihood at the begining.
 - > err : A empty string if success, otherwise the error message as a char (lik = 1e10 in this case).
- y : Observation vector. A nObs x T double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodDSGE](#), [nb_kalmanSmoothenTVPDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_kalmanLikelihoodUnivariateBreakPointDSGE**

lik = nb_kalmanLikelihoodUnivariateBreakPointDSGE(par, model, y, varargin)

Description:

Diffuse univariate piecewise linear Kalman filter. With potentially missing observations!

The model function must return the given matrices in the system below:

Observation equation:

$$y = H \cdot x(t)$$

State equation:

$$x = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.

- model : A function handle returning the following matrices (in this order):

```
> H      : Observation matrix. As a nObs x nEndo double.  
> A      : State transition matrix. A cell with size 1 x nStates. Each element as a nEndo x nEndo double.  
> C      : The impact of shock matrix. A cell with size 1 x nStates. Each element as a nEndo x nExo double.  
> x      : Initial state vector. As a nEndo x 1 double.  
> P      : Initial state variance. As a nStationaryVar x nStationaryVar double.  
> Pinf   : Initial state variance of the non-stationary variables. As a nNonStationaryVar x nNonStationaryVar double.  
> options : A struct with the fields:  
           * kf_riccatiTol : Converge criteria on the Kalman gain.
```

```

* kf_presample : Discarded observation of the
    likelihood at the begining.

* states       : A nobs vector with the states
    at the given period.

* ss           : The steady-state of the model in
    each regime. As a cell array.

> err         : A empty string if success, otherwise the error
    message as a char (lik = le10 in this case).

- y : Observation vector. A nObs x T double.

```

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodBreakPointDSGE](#), [nb_setUpForDiffuseFilter](#)
[nb_kalmanSmotherUnivariateBreakPointDSGE](#)
[nb_kalmanLikelihoodDiffuseBreakPointDSGE](#)

Written by Kenneth Åkerhagen Paulsen

◆ **nb_kalmanLikelihoodUnivariateDSGE**

lik = nb_kalmanLikelihoodUnivariateDSGE(par,model,y,varargin)

Description:

Diffuse univariate linear Kalman filter. With potentially missing observations!

The model function must return the given matrices in the system below:

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A*x(t-1) + C*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance Q

See for example: Koopman and Durbin (1998), "Fast filtering ans smoothing for multivariate state space models".

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. As a nEndo x nEndo double.
 - > C : The impact of shock matrix. As a nEndo x nExo double.
 - > x : Initial state vector. As a nEndo x 1 double.
 - > P : Initial state variance. As a nStationaryVar x nStationaryVar double.
 - > Pinf : Initial state variance of the non-stationary variables. As a nNonStationaryVar x nNonStationaryVar double.
 - > options : A struct with the fields:
 - * kf_riccatiTol : Converge criteria on the Kalman gain.
 - * kf_presample : Discarded observation of the likelihood at the begining.
- y : Observation vector. A nObs x T double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodDSGE](#), [nb_kalmanSmootherUnivariateDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanLikelihoodDiffuseDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_kalmanLikelihoodUnivariateStochasticTrendDSGE

lik = nb_kalmanLikelihoodUnivariateStochasticTrendDSGE(par,estStruct)

Description:

Diffuse univariate piecewise linear Kalman filter with updating of the "steady state" based on some stochastic trends of the observation part of the model. Will also handle potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + B(t) \cdot z(t) + C(t) \cdot u(t)$$

where $ss(t) = G(theta, x(t-1))$ and $A(t)$, $B(t)$ and $C(t)$ are the solution around this updated approximation point (or steady state).

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : Current vector of the estimated parameters.
- estStruct : See the `nb_dsg.e.getObjectiveForEstimation` method.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.
- A : The updated estimates of the transition matrix.
- B : The updated estimates of the contant term of the state equation.
- C : The updated estimates of the shock impact matrix.
- ss : The updated estimates of the approximation point (steady state).
- p : The updated estimates of the parameters of the model.

See also:

[nb_kalmanSmootherUnivariateStochasticTrendDSGE](#), [nb_dsg.e.updateSolution](#)
[nb_kalmanLikelihoodDiffuseStochasticTrendDSGE](#), [nb_setUpForDiffuseFilter](#)

Written by Kenneth Sæterhagen Paulsen.

◆ **nb_kalmanLikelihoodUnivariateTVPDSGE**

`lik = nb_kalmanLikelihoodUnivariateTVPDSGE(par, model, y, varargin)`

Description:

Diffuse univariate Kalman filter with observed time-varying parameters that does not affect the steady state.

The model function must return the given matrices in the system below:

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A(t) \cdot x(t-1) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > H : Observation matrix. As a nObs x nEndo double.
 - > A : State transition matrix. As a nEndo x nEndo x T double.
 - > C : The impact of shock matrix. As a nEndo x nExo x T double.
 - > x : Initial state vector. As a nEndo x 1 double.
 - > P : Initial state variance. As a nStationaryVar x nStationaryVar double.
 - > Pinf : Initial state variance of the non-stationary variables. As a nNonStationaryVar x nNonStationaryVar double.
 - > options : A struct with the fields:
 - * kf_riccatiTol : Converge criteria on the Kalman gain.
 - * kf_presample : Discarded observation of the likelihood at the begining.
 - > err : A empty string if success, otherwise the error message as a char (lik = 1e10 in this case).
- y : Observation vector. A nObs x T double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanLikelihoodTVPDSGE](#), [nb_kalmanSmootherUnivariateTVPDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanLikelihoodDiffuseTVPDSGE](#)

Written by Kenneth Åkerhagen Paulsen

◆ **nb_kalmanSmootherAndLikelihood**

```
[xf,xu,xs,us,lik,Ps,Ps_1,Pf,Pu] = nb_kalmanSmootherAndLikelihood(H,R,...  
A,B,obs,x0,P0,y,kalmanTol,kf_presample,G,z,s)
```

Description:

Kalman smoother. With potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t) + v(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + B \cdot u(t)$$

or if G and z is given

$$x(t) = A \cdot x(t-1) + G \cdot z(t) + s(t) \cdot B \cdot u(t)$$

Where $u \sim N(0, I)$ and $v \sim N(0, R)$.

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models" and Hamilton (1994), "Time Series Analysis"

Input:

- H : Observation matrix. As a nObs x nEndo double.
- R : Measurement error covariance matrix. As a nObs x nObs double. If you assume no measurement error, give zeros(nObs).
- A : State transition matrix. As a size nEndo x nEndo.
- B : The impact of shock matrix. As a size nEndo x nExo.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Kalman tolerance.
- kf_presample : Number of periods before the likelihood is calculated.

- G : A nEndo x nExo double.
- z : A nExo x T double
- s : A 1 x T double. With the stochastic-volatility process.
Defaults to 1.

Output:

- xf : The filtered estimates ($x_{t+1|t}$) of the x in the equation above.
A nEndo x T + 1 double. $xf(:,1)$ is set to x_0 , and is interpreted as $x_{1|0}$.
- xu : The updated estimates ($x_{t|t}$) of the x in the equation above.
A nEndo x T + 1 double. $xu(:,1)$ is set to x_0 , and is interpreted as $x_{0|0}$.
- xs : The smoothed estimates ($x_{t|T}$) of the x in the equation above.
A nEndo x T double. $xs(:,1)$ is interpreted as $x_{1|T}$.
- us : The smoothed estimates of the u in the equation above.
A nExo x T double.
- lik : Minus the log likelihood. As a 1 x 1 double.
- Ps : Smoothed one step ahead covariance matrices. $P_{t|T}$.
A nEndo x nEndo x T double. $Ps(:,1)$ is interpreted as $P_{1|T}$.
- Ps_1 : Smoothed estimate of $\text{Cov}(x(t) | x(t-1) | T)$. $P_{-1|T}$.
A nEndo x nEndo x T double.
- Pf : Filtered one step ahead covariance matrices. $P_{t+1|t}$.
A nEndo x nEndo x T+1 double. P0 at first page.
- Pu : Updated one step ahead covariance matrices. $P_{t|t}$.
A nEndo x nEndo x T+1 double. P0 at first page.

See also:

[nb_kalmanLikelihoodMissingDSGE](#), [nb_kalmanSmoothenAndLikelihoodUnivariate](#)

Written by Kenneth Åkerhagen Paulsen.

◆ **nb_kalmanSmoothenAndLikelihoodUnivariate**

```
[xf,xu,xs,us,lik,Ps,Ps_1,Pf,Pinf,Pu,Pinfu] = ...
    nb_kalmanSmoothenAndLikelihood(H,R,A,B,obs,x0,P0,Pinf0,y, ...
        kalmanTol,kf_presample,G,z)
```

Description:

Kalman smoother. With potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t) + v(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + B \cdot u(t)$$

or if G and z is given

$$x(t) = A \cdot x(t-1) + G \cdot z(t) + B \cdot u(t)$$

Where $u \sim N(0, I)$ and $v \sim N(0, R)$.

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models" and Hamilton (1994), "Time Series Analysis"

Input:

- H : Observation matrix. As a nObs x nEndo double.
- R : Measurement error covariance matrix. As a nObs x nObs double. If you assume no measurement error, give zeros(nObs).
- A : State transition matrix. As a size nEndo x nEndo.
- B : The impact of shock matrix. As a size nEndo x nExo.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- Pinfo : Diffuse initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Kalman tolerance.
- kf_psample : Number of periods before the likelihood is calculated.
- G : A nEndo x nExo double.
- z : A nExo x T double

Output:

- xf : The filtered estimates ($x_{t+1|t}$) of the x in the equation above. A nEndo x T + 1 double. xf(:,1) is set to x0, and is interpreted as $x_{1|0}$.
- xu : The updated estimates ($x_{t|t}$) of the x in the equation above. A nEndo x T + 1 double. xu(:,1) is set to x0, and is interpreted as $x_{0|0}$.
- xs : The smoothed estimates ($x_{t|T}$) of the x in the equation above. A nEndo x T + 1 double. xs(:,1) is interpreted as $x_{1|T}$.

- us : The smoothed estimates of the u in the equation above.
A nExo x T double.
- lik : Minus the log likelihood. As a 1 x 1 double.
- Ps : Smoothed one step ahead covariance matrices. P t|T.
A nEndo x nEndo x T double. Ps(:,1) is interpreted as
P 1|T.
- Ps_1 : Smoothed estimate of Cov(x(t) x(t-1)|T). P_1 t|T.
A nEndo x nEndo x T double.
- Pf : Filtered one step ahead covariance matrices. P t+1|t.
A nEndo x nEndo x T+1 double. P0 at first page.
- Pinf : Filtered one step ahead covariance matrices of diffuse steps.
Pinf t+1|t. A nEndo x nEndo x S double, where S is the
number of periods in the diffuse stage. Pinf0 at first page.
- Pu : Updated one step ahead covariance matrices. P t|t.
A nEndo x nEndo x T+1 double. P0 at first page.
- Pinfu : Filtered one step ahead covariance matrices of diffuse steps.
Pinfu t+1|t. A nEndo x nEndo x S double, where S is the
number of periods in the diffuse stage. Pinfu0 at first page.

See also:

[nb_kalmanLikelihoodMissingDSGE](#), [nb_kalmanSmoothenAndLikelihood](#)

Written by Kenneth Åkerhagen Paulsen.

◆ **nb_kalmanSmoothenBreakPointDSGE**

```
[xf, xs, us, xu, uu] = nb_kalmanSmoothenBreakPointDSGE(H, A, C, ss, obs, x0, P0, ...
y, kalmanTol, states)
```

Description:

Piecewise linear Kalman smoother. With potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t) + v(t)$$

State equation:

$$x(t) = ss\{s(t)\} + A\{s(t)\} \cdot (x(t-1) - ss\{s(t)\}) + C\{s(t)\} \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. A cell with size 1 x nStates.
Each element as a nEndo x nEndo double.

- C : The impact of shock matrix. A cell with size 1 x nStates.
Each element as a nEndo x nExo double.
- ss : Steady-state of the endogenous variables. A cell with size 1 x nStates. Each element as a nEndo x 1 double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.
- states : A 1 x nObs double with the break period to use for each period, i.e. how to index A, C and ss at each period.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanSmoothenDSGE](#), [nb_kalmanLikelihoodBreakPointDSGE](#)

Written by Kenneth Åkerhagen Paulsen.

◆ **nb_kalmanSmoothenDSGE**

[xf, xs, us, xu, uu] = nb_kalmanSmoothenDSGE(H, A, B, obs, x0, P0, y, kalmanTol)

Description:

Kalman smoother. No missing observations!

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + B \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. As a nEndo x nEndo double.
- B : The impact of shock matrix. As a nEndo x nExo double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Kalman tolerance.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_kalmanSmoothenDiffuseBreakPointDSGE

```
[xf,xs,us,xu,uu] = nb_kalmanSmoothenDiffuseBreakPointDSGE(H,A,C,ss,obs,...  
x0,P0,P0INF,y,kalmanTol,states)
```

Description:

Diffuse piecewise linear Kalman smoother. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t) + v(t)$

State equation:
 $x(t) = ss(t) + A(t)*(x(t-1) - ss(t)) + C(t)*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. A cell with size 1 x nStates.
Each element as a nEndo x nEndo double.
- C : The impact of shock matrix. A cell with size 1 x nStates.
Each element as a nEndo x nExo double.
- ss : Steady-state of the endogenous variables. A cell with size 1 x nStates. Each element as a nEndo x 1 double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables.
As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.
- states : A 1 x nobs double with the break period to use for each period, i.e. how to index A,B and ss at each period.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodDiffuseBreakPointDSGE](#), [nb_kalmanSmoothenBreakPointDSGE](#)
[nb_setUpForDiffuseFilter](#)

Written by Kenneth Åkerhagen Paulsen.

◆ nb_kalmanSmoothenDiffuseDSGE

```
[xf,xs,us,xu,uu] = nb_kalmanSmoothenDiffuseDSGE(H,A,C,obs,x0,P0S,P0INF,...  
y,kalmanTol)
```

Description:

Diffuse Kalman smoother to handle non-stationary models. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A*x(t-1) + C*u(t)$

Where $u \sim N(0, I)$, meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. As a nEndo x nEndo double.
- C : The impact of shock matrix. As a nEndo x nExo double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanSmoothenDSGE](#), [nb_kalmanLikelihoodDiffuseDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanSmoothenUnivariateDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_kalmanSmoothenDiffuseStochasticTrendDSGE**

```
[xf,xs,us,xu,uu] = nb_kalmanSmootherDiffuseStochasticTrendDSGE(H, ...
    solution,options,results,obs,x0,P0,P0INF,y, ...
    kalmanTol)
```

Description:

Diffuse piecewise linear Kalman smoother with updating of the "steady state" based on some stochastic trends of the observation part of the model. Will also handle potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = ss(t) + A(t)*(x(t-1) - ss(t)) + B(t)*z(t) + C(t)*u(t)$

where $ss(t) = G(theta, x(t-1))$ and $A(t)$, $B(t)$ and $C(t)$ are the solution around this updated approximation point (or steady state).

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- solution : A struct containing the initial solution of the model. See nb_dsge.solution property for more on this input.
- options : A struct with the model options. See the nb_dsge.createEstOptionsStruct method for more on this input.
- results : A struct storing the estimation or calibration of the model. See nb_dsge.results property for more on this input.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.

Output:

- xf : The filtered estimates of the x in the equation above. As a T x nEndo double.
- xs : The smoothed estimates of the x in the equation above. As a T x nEndo double.

- us : The smoothed estimates of the u in the equation above. As a T x nExo double.
- xu : The updated estimates of the x in the equation above. As a T x nEndo double.
- uu : The updated estimates of the u in the equation above. As a T x nExo double.
- A : The updated estimates of the transition matrix.
A nEndo x nEndo x T + 1 double, starting from A 0|0 to A T|T.
- B : The updated estimates of the contant term of the state equation.
A nEndo x nDet x T + 1 double, starting from B 0|0 to B T|T.
nDet are either 0 or 1.
- C : The updated estimates of the shock impact matrix.
A nEndo x nExo x T + 1 double, starting from C 0|0 to C T|T.
- ss : The updated estimates of the approximation point (steady state).
A nEndo x 1 x T + 1 double, starting from ss 0|0 to ss T|T.
- p : The updated estimates of the parameters of the model. As a T x nParam double.

See also:

[nb_kalmanLikelihoodDiffuseStochasticTrendDSGE](#), [nb_dsgc.updateSolution](#)
[nb_kalmanSmoothenUnivariateStochasticTrendDSGE](#), [nb_setUpForDiffuseFilter](#)

Written by Kenneth Sæterhagen Paulsen.

◆ **nb_kalmanSmoothenDiffuseTVPDSGE**

```
[xf, xs, us] = nb_kalmanSmoothenDiffuseTVPDSGE(H, A, C, obs, x0, P0, P0INF, y, ...
                                                 kalmanTol)
```

Description:

Diffuse Kalman smoother with observed time-varying paramerters that does not affect the steady-state. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A(t)*x(t-1) + B(t)*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.

- A : State transition matrix. With size nEndo x nEndo x nPeriods.
- C : The impact of shock matrix. With size nEndo x nExo x nPeriods.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodDiffuseTVPDSGE](#), [nb_kalmanSmootherTVPDSGE](#)
[nb_setUpForDiffuseFilter](#)

Written by Kenneth Sæterhagen Paulsen.

◆ nb_kalmanSmootherMissingDSGE

```
[xf, xs, us, xu, uu] = nb_kalmanSmootherMissingDSGE(H, A, C, obs, x0, P0, y, ...
kalmanTol)
```

Description:

Kalman smoother. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A*x(t-1) + Bu(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. As a size nEndo x nEndo.
- C : The impact of shock matrix. As a size nEndo x nExo.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Kalman tolerance.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodMissingDSGE](#)

Written by Kenneth Sæterhagen Paulsen.

◆ nb_kalmanSmoothenTVPSGE

```
[xf, xs, us] = nb_kalmanSmoothenTVPSGE(H, A, B, obs, x0, P0, y, ...
                                         kalmanTol)
```

Description:

Kalman smoother with observed time-varying parameters that does not affect the steady-state. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A(t)*x(t-1) + C(t)*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. As a size nEndo x nEndo x nPeriods.
- C : The impact of shock matrix. As a size nEndo x nExo x nPeriods.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.

See also:

[nb_kalmanSmoothenDSGE](#), [nb_kalmanLikelihoodTVPDSGE](#)

Written by Kenneth Sæterhagen Paulsen.

◆ nb_kalmanSmoothenUnivariateBreakPointDSGE

```
[xf,xs,us,xu,uu] = nb_kalmanSmoothenUnivariateBreakPointDSGE(H,A,C,ss,...  
obs,x0,P0,P0INF,y,kalmanTol,states)
```

Description:

Diffuse univariate piecewise linear Kalman smoother. With potentially missing observations!

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. A cell with size 1 x nStates.
Each element as a nEndo x nEndo double.
- C : The impact of shock matrix. A cell with size 1 x nStates.
Each element as a nEndo x nExo double.
- ss : Steady-state of the endogenous variables. A cell with size 1 x nStates. Each element as a nEndo x 1 double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables.
As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.
- states : A 1 x nobs double with the break period to use for each period, i.e. how to index A,B and ss at each period.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodUnivariateBreakPointDSGE](#)
[nb_kalmanSmoothenBreakPointDSGE](#), [nb_setUpForDiffuseFilter](#)

Written by Kenneth Sæterhagen Paulsen.

◆ **nb_kalmanSmoothenUnivariateDSGE**

```
[xf,xs,us,xu,uu] = nb_kalmanSmoothenUnivariateDSGE(H,A,C,obs,x0,P0S,...  
P0INF,y,kalmanTol)
```

Description:

Diffuse univariate Kalman smoother to handle non-stationary models. With potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A*x(t-1) + C*u(t)$

Where $u \sim N(0, I)$, meaning u is gaussian noise with covariance I

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. As a nEndo x nEndo double.
- C : The impact of shock matrix. As a nEndo x nExo double.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.
- xu : The updated estimates of the x in the equation above.
- uu : The updated estimates of the u in the equation above.

See also:

[nb_kalmanSmoothenDSGE](#), [nb_kalmanLikelihoodUnivariateDSGE](#)
[nb_setUpForDiffuseFilter](#), [nb_kalmanSmoothenDiffuseDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_kalmanSmoothenUnivariateStochasticTrendDSGE**

```
[xf,xs,us,xu,uu,A,B,C,ss] = ...
nb_kalmanSmoothenUnivariateStochasticTrendDSGE(H,solution,options, ...
results,obs,x0,P0,P0INF,y,kalmanTol)
```

Description:

Diffuse univariate piecewise linear Kalman smoother with updating of the "steady state" based on some stochastic trends of the observation part of the model. Will also handle potentially missing observations!

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = ss(t) + A(t)*(x(t-1) - ss(t)) + B(t)*z(t) + C(t)*u(t)$

where $ss(t) = G(theta, x(t-1))$ and $A(t)$, $B(t)$ and $C(t)$ are the solution around this updated approximation point (or steady state).

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- solution : A struct containing the initial solution of the model. See nb_dsg.e.solution property for more on this input.
- options : A struct with the model options. See the nb_dsg.e.createEstOptionsStruct method for more on this input.
- results : A struct storing the estimation or calibration of the model. See nb_dsg.e.results property for more on this input.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.

Output:

- xf : The filtered estimates of the x in the equation above. As a T x nEndo double.
- xs : The smoothed estimates of the x in the equation above. As a T x nEndo double.

- us : The smoothed estimates of the u in the equation above. As a T x nExo double.
- xu : The updated estimates of the x in the equation above. As a T x nEndo double.
- uu : The updated estimates of the u in the equation above. As a T x nExo double.
- A : The updated estimates of the transition matrix.
A nEndo x nEndo x T + 1 double, starting from A 0|0 to A T|T.
- B : The updated estimates of the contant term of the state equation.
A nEndo x nDet x T + 1 double, starting from B 0|0 to B T|T.
nDet are either 0 or 1.
- C : The updated estimates of the shock impact matrix.
A nEndo x nExo x T + 1 double, starting from C 0|0 to C T|T.
- ss : The updated estimates of the approximation point (steady state).
A nEndo x 1 x T + 1 double, starting from ss 0|0 to ss T|T.
- p : The updated estimates of the parameters of the model. As a T x nParam double.

See also:

[nb_kalmanLikelihoodUnivariateStochasticTrendDSGE](#)
[nb_kalmanSmoothenDiffuseStochasticTrendDSGE](#), [nb_setUpForDiffuseFilter](#)

Written by Kenneth Å!terhagen Paulsen.

◆ **nb_kalmanSmoothenUnivariateTVPDSGE**

```
[xf,xs,us] = nb_kalmanSmoothenUnivariateTVPDSGE(H,A,C,obs,x0,P0,...  
P0INF,y,kalmanTol)
```

Description:

Diffuse univariate Kalman smoother with observed time-varying
paramerters that does not affect the steady-state. With potentially
missing observations!

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A(t) \cdot x(t-1) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I

Input:

- H : Observation matrix. As a nObs x nEndo double.
- A : State transition matrix. With size nEndo x nEndo x nPeriods.
- C : The impact of shock matrix. With size nEndo x nExo x nPeriods.
- obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.
- x0 : Initial state vector. As a nEndo x 1 double.
- P0 : Initial state variance. As a nEndo x nEndo double.
- P0INF : Initial state variance of the non-stationary variables. As a nEndo x nEndo double.
- y : Observation vector. A nObs x T double.
- kalmanTol : Tolerance level for rcond of the forecast variance.

Output:

- xf : The filtered estimates of the x in the equation above.
- xs : The smoothed estimates of the x in the equation above.
- us : The smoothed estimates of the u in the equation above.

See also:

[nb_kalmanLikelihoodDiffuseTVPDSGE](#), [nb_kalmanSmotherTVPDSGE](#)
[nb_setUpForDiffuseFilter](#)

Written by Kenneth Åkerhagen Paulsen.

◆ **nb_kalmanfilter**

[x,u,v] = nb_kalmanfilter(model,y,z,varargin)

Description:

The model function must return the given matrices in the system below (I.e. d,H,R,T,c,A,B,Q,G) :

Observation equation:
 $y = d + Hx + Gz + v$

State equation:
 $x = c + Ax_{-1} + Tz + Bu$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q
 $v \sim N(0, R)$ meaning v is gaussian noise with covariance R

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- model : A function handle returning the following matrices (in this order) :

```
> x      : Initial state vector.  
> P      : Initial state variance.  
> d      : Constant in the observation equation.  
> H      : Observation matrix.  
> R      : Measurement noise covariance.  
> T      : Matrix on exogenous variables in the observation  
          equation. Set it to 0 if not needed!  
> c      : Constant in the state equation.  
> A      : State transition matrix.  
> Q      : Process noise covariance.  
> B      : Input matrix, optional.  
> G      : Matrix on exogenous variables in the state  
          equation. Set it to 0 if not needed!  
> u      : Input control vector, optional.  
> obs    : Index of observables in the state vector.  
> failed : Give true if model cannot be solved. lik will  
          be returned a value of 1e10 in this case.
```

- y : Observation vector. A nvar x nobs double.

- z : Observation vector of exogenous. A nxvar x nobs double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- x : The filtered estimates of the x in the equation above. (x t|t)
- u : The filtered estimates of the u in the equation above. (u t|t)
- v : The filtered estimates of the v in the equation above. (v t|t)

See also:

[nb_kalmansmooth](#), [nb_kalmanlikelihood](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_kalmanlikelihood**

lik = nb_kalmanlikelihood(par,model,y,z,start,varargin)

Description:

The model function must return the given matrices in the system below (I.e. d,H,R,T,c,A,B,Q,G) :

Observation equation:

$$y(t) = d + H*x(t) + T*z(t) + v(t)$$

State equation:

$$x(t) = c + A*x(t-1) + G*z(t) + B*u(t)$$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q
 $v \sim N(0, R)$ meaning v is gaussian noise with covariance R

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.
- model : A function handle returning the following matrices (in this order):
 - > x : Initial state vector.
 - > P : Initial state variance.
 - > d : Constant in the observation equation.
 - > H : Observation matrix.
 - > R : Measurement noise covariance.
 - > T : Matrix on exogenous variables in the observation equation. Set it to 0 if not needed!
 - > c : Constant in the state equation.
 - > A : State transition matrix.
 - > Q : Process noise covariance.
 - > B : Input matrix, optional.
 - > G : Matrix on exogenous variables in the state equation. Set it to 0 if not needed!
 - > u : Input control vector, optional.
 - > obs : Index of observables in the state vector.
 - > failed : Give true if model cannot be solved. lik = 1e10 in this case.
- y : Observation vector. A nvar x nobs double.
- z : Observation vector of exogenous. A nxvar x nobs double.
- start : Start observation of the likelihood calculation.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

nb_kalmanlikelihood_missing, nb_kalmansmooth

Written by Kenneth Sæterhagen Paulsen

◆ nb_kalmanlikelihood_missing

```
lik = nb_kalmanlikelihood_missing(par,model,y,z,varargin)
```

Description:

The model function must return the given matrices in the system below (I.e. d,H,R,T,c,A,B,Q,G) :

Observation equation:

$$y(t) = d + H*x(t) + T*z(t) + v(t)$$

State equation:

$$x(t) = c + A*x(t-1) + G*z(t) + B*u(t)$$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q
 $v \sim N(0, R)$ meaning v is gaussian noise with covariance R

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- par : A parameter vector which is given as the first input to the function handle given by the model input.

- model : A function handle returning the following matrices (in this order):

```
> x      : Initial state vector.  
> P      : Initial state variance.  
> d      : Constant in the observation equation.  
> H      : Observation matrix.  
> R      : Measurement noise covariance.  
> T      : Matrix on exogenous variables in the observation equation. Set it to 0 if not needed!  
> c      : Constant in the state equation.  
> A      : State transition matrix.  
> Q      : Process noise covariance.  
> B      : Input matrix, optional.  
> G      : Matrix on exogenous variables in the state equation. Set it to 0 if not needed!  
> u      : Input control vector, optional.  
> obs    : Index of observables in the state vector.  
> failed : Give true if model cannot be solved. lik = 1e10 in this case.
```

- y : Observation vector. A nvar x nobs double.

- z : Observation vector of exogenous. A nxvar x nobs double.

- start : Start observation of the likelihood calculation.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- lik : Minus the log likelihood. As a 1 x 1 double.

See also:

[nb_kalmanlikelihood](#), [nb_kalmansmootherr_missing](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_kalmansmootherr

[xf, xs, us, vs, xu] = nb_kalmansmootherr(model, y, z, varargin)

Description:

Kalman smoother. No missing observations!

The model function must return the given matrices in the system below (I.e. d, H, R, T, c, A, B, Q, G) :

Observation equation:

$$y(t) = d + H*x(t) + T*z(t) + v(t)$$

State equation:

$$x(t) = c + A*x(t-1) + G*z(t) + B*u(t)$$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q
 $v \sim N(0, R)$ meaning v is gaussian noise with covariance R

See for example: Koopman and Durbin (1998), "Fast filtering and smoothing for multivariate state space models".

Input:

- model : A function handle returning the following matrices (in this order):

> x	: Initial state vector.
> P	: Initial state variance.
> d	: Constant in the observation equation.
> H	: Observation matrix.
> R	: Measurement noise covariance.
> T	: Matrix on exogenous variables in the observation equation. Set it to 0 if not needed!
> c	: Constant in the state equation.
> A	: State transition matrix.
> Q	: Process noise covariance.
> B	: Input matrix, optional.

```

> G      : Matrix on exogenous variables in the state
           equation. Set it to 0 if not needed!
> u      : Input control vector, optional.
> obs    : Index of observables in the state vector.
> failed : Give true if model cannot be solved. lik will
           be returned a value of 1e10 in this case.

- y   : Observation vector. A nvar x nobs double.

- z   : Observation vector of exogenous. A nxvar x nobs double.

```

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- xf : The filtered estimates of the x in the equation above. ($x_{t+1|t}$)
- xs : The smoothed estimates of the x in the equation above. ($x_{t|T}$)
- us : The smoothed estimates of the u in the equation above. ($u_{t|T}$)
- vs : The smoothed estimates of the v in the equation above. ($v_{t|T}$)
- xu : The updated estimates of the x in the equation above. ($x_{t|t}$)

See also:

[nb_kalmansmooth_missing](#), [nb_kalmanlikelihood](#)

Written by Kenneth Sæterhagen Paulsen

◆ [nb_kalmansmooth_missing](#)

[xf, xs, us, vs, xu, uu, Ps] = nb_kalmansmooth_missing(par, model, y, z, varargin)

Description:

Kalman smoother. With potentially missing observations!

The model function must return the given matrices in the system below (I.e. d, H, R, T, c, A, B, Q, G) :

Observation equation:

$$y(t) = d + H \cdot x(t) + T \cdot z(t) + v(t)$$

State equation:

$$x(t) = c + A \cdot x(t-1) + G \cdot z(t) + B \cdot u(t)$$

Where $u \sim N(0, Q)$ meaning u is gaussian noise with covariance Q
 $v \sim N(0, R)$ meaning v is gaussian noise with covariance R

See for example: Koopman and Durbin (1998), "Fast filtering ans smoothing for multivariate state space models".

Input:

- model : A function handle returning the following matrices (in this order):

- > x0 : Initial state vector. As a nvar x 1 double.
- > P0 : Initial state variance. As a nvar x nvar double.
- > d : Constant in the observation equation.
- > H : Observation matrix (defaults to identity).
- > R : Measurement noise covariance (required).
- > c : Constant in the state equation.
- > A : State transition matrix (defaults to identity).
- > Q : Process noise covariance (defaults to zero).
- > B : Input matrix, optional (defaults to zero).
- > u : Input control vector, optional (defaults to zero).
- > obs : Index of observables in the vector of state variables. A logical vector with size size(A,1) x 1.

- y : Observation vector. A nvar x nobs double.

- z : Observation vector of exogenous. A nxvar x nobs double.

Optional input:

- varargin : Optional inputs given to the function handle given by the model input.

Output:

- xf : The filtered estimates of the x in the equation above. (x t+1|t)
- xs : The smoothed estimates of the x in the equation above. (x t|T)
- us : The smoothed estimates of the u in the equation above. (u t|T)
- vs : The smoothed estimates of the v in the equation above. (v t|T)
- xu : The updated estimates of the x in the equation above. (x t|t)
- uu : The updated estimates of the u in the equation above. (u t|t)
- Ps : Smoothed one step ahead covariance matrices. P t|T. Ps(:,1) is interpreted as P 1|T.

See also:

[nb_kalmansmoothes](#), [nb_kalmanlikelihood_missing](#)

◆ nb_lasso

```
[beta,exitflag,residual,X] = nb_lasso(y,X,t,constant,options)
```

Description:

LASSO estimation using the local linearization and active set method proposed in [Osborne et al., 2000], [Osborne et al., 2000b].

Solves the constrained optimization problem:

```
min (y - X*beta)^2  
s.t. sum(abs(beta)) <= 1/t
```

The code is an adaption of code written by M. Schmidt. See:

- <https://www.cs.ubc.ca/~schmidtm/Software/lasso.html>
- M. Schmidt. Least Squares Optimization with L1-Norm Regularization. CS542B Project Report, 2005.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- X : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression.
- t : The value of t in the minimization problem. Default is inf. Must be a scalar number greater or equal to 0.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false. The constant term is first estimate by the mean of y, and then y is demeaned before doing the LASSO estimation.
- options : Optimization options. As a struct. See nb_lasso.optimset.

Output:

- beta : A (extra + nxvar) x neq matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- exitflag : The reason for exiting. One of:
 - > -1 : Numerical breakdown, check for dependent columns.
 - > -2 : Maximum number of iteration reached (maxIter).

See also: nb_interpretExitFlag. Set type to 'nb_lasso'.

- residual : Residual from the estimated equation. As an nobs x neq matrix.

- X : Regressors including constant. If more than one equation this will be given as `kron(I,x)`.

See also:

[nb_lasso.optimset](#), [nb_ols](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_legendrePoly**

`Q = nb_legendrePoly(P,K,E)`

Description:

Implementation of the Legendre lag polynominal.

Input:

- P : Number of lags of the polynomial. If empty it will default to K. Either as a scalar integer or a vector of integers with length E.
- K : Number of lags of the regressor. Either as a scalar integer or a vector of integers with length E.
- E : Number of variables.

Output:

- Q : A $P \times E$ x $K \times E$ (or $\text{sum}(P(i)) \times \text{sum}(K(i))$) double with the mapping matrix.

See also:

[nb_midasFunc](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_midasBootstrap**

`ysim = nb_midasBootstrap(p,resid,x,constant,func,nExo,AR,draws,method)`

Description:

Calculate residuals of the MIDAS model given a parameter set.

Input:

- p : A nParam x 1 double vector.
- resid : A nobs x 1 double with the residuals of the model.
- x : A nobs x nExo*nLags double with the exogenous variables of the model. Except constant.
- constant : true or false. true if constant is included in the model.
- func : A function handle that map from the non-linear parameters to the linear once. If empty the MIDAS model is unrestricted.
- nExo : Number of exogenous variables of the model, excluding lag and constant.
- AR : true or false. true if AR specification of the MIDAS model is used.
- draws : Number of draws from the distribution of the dependent variable.
- method : See the method input to the nb_bootstrap function. Default is 'bootstrap'.

Output:

- ysim : A nobs x draws double with the simulated dependent variable of the model.

See also:

[nb_midasMapToLinear](#), [nb_midasFunc](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_midasFunc**

```
beta = nb_midasFunc(y,x)
[beta, stdBeta, tStatBeta, pValBeta, residual, sigma, betaD, sigmaD] = nb_midasFunc(y,x, ...
    constant, type, nSteps, stdType, nExo, nLags, varargin)
```

Description:

Estimate MIDAS model.

$y = x*f(beta) + \epsilon$, $\epsilon \sim N(0, \epsilon^* \epsilon / nobs)$ (1)

Input:

- y : A double matrix of size nobs x 1 of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the

regression. These are of a higher frequency than y. Lags of regressors should already be added. The nExo input must be set.

The order of the regressors must be;
var1_lag1, var2_lag1, var1_lag2, var2_lag2, var1_lag3
i.e. they can have different number of lags. See the nLags input!

- constant : true or false. If true a constant is included.
- AR : true or false. Indicate that you are estimating a AR specification of the MIDAS model.
- type : The type of MIDAS model to estimate. Either; 'unrestricted', 'beta', 'legendre', 'mean' or 'almon'. If set to 'beta' profiling is used, otherwise OLS is used.
- nSteps : Add leads to make the MIDAS models able to produce direct forecast longer than 1 period forward. Default is 1. Must be an integer larger than 0.
- stdType :
 - 'w' : Will return White heteroskedasticity robust standard errors.
 - 'nw' : Will return Newey-West heteroskedasticity and autocorrelation robust standard errors.
- To estimate the covariance matrix of the estimated parameters the bartlett kernel is used with the bandwidth set to $\text{floor}(4(T/100)^{(2/9)})$ as Eviews also uses.
- 'h' : Will return homoskedasticity only robust standard errors. Default.
- nExo : Number of exogenous variables excluding the lags.
- nLags : Either a scalar integer with the number of lags of all the regressors, or a vector of integers with length nExo with the number of lags of each regressor. Default is 1!

Optional inputs:

- 'draws' : Number of draws from the parameter distribution when bootstrapping the standard error of the parameters.
- 'polyLags' : The number of polynomial lags to use when type is set to 'legendre' or 'almon'. Either as a scalar integer or a vector of integers with length nExo.

Output:

- beta : A nxvar x nSteps matrix with the estimated parameters.
- stdBeta : A nxvar x nSteps matrix with the standard deviation of the estimated parameters.
- tStatBeta : A nxvar x nSteps matrix with t-statistics of the

estimated paramteres.

- pValBeta : A nxvar x nSteps matrix with the p-values of the estimated paramteres.
- residual : Residual from the estimated equation. As an nobs x nSteps matrix.
- sigma : Covariance matrix of estimated residuals. Will be a diagonal matrix with size nSteps x nSteps.
- betaD : The bootstrapped parameters of the model. As a nxvar x nSteps x draws double.
- sigmaD : The bootstrapped covariances of the residuals of the model. Assumed to be diagonal. As a nSteps x nSteps x draws double.

See also

`nb_midasEstimator`, `nb_midas`

Written by Kenneth S. Paulsen

◆ **`nb_midasMapToLinear`**

```
beta = nb_midasMapToLinear(p,AR,constant,nExo,nLags)
```

Description:

Map from the non-linear estimated parameter to the linear coefficient of the MIDAS model, i.e. map from

$y = X \cdot f(p) + \epsilon$, $\epsilon \sim N(0, \epsilon' \cdot \epsilon / nobs)$ (1)

to

$y = X \cdot \beta + \epsilon$, $\epsilon \sim N(0, \epsilon' \cdot \epsilon / nobs)$ (1)

Input:

- p : A nParam x 1 double vector.
- func : A function handle that map from the non-linear parameters to the linear once.
- AR : true or false. true if AR specification of the MIDAS model is used.
- constant : true or false. true if constant is included in the model.
- nExo : Number of exogenous variables of the model, exluding lag and constant.
- nLags : Number of lags of the exogenous variables.

Output:

- beta : A constant + AR + nExo*nLags x 1 double vector.

See also:

[nb_midasFunc](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_midasResiduals**

resid = nb_midasResiduals(p,y,x,constant,func,nExo,AR)

Description:

Calculate residuals of the MIDAS model given a parameter set.

Input:

- p : A nParam x 1 double vector.
- y : A nobs x 1 double with the dependent variable of the model.
- x : A nobs x nExo*nLags double with the exogenous variables of the model. Except constant.
- constant : true or false. true if constant is included in the model.
- func : A function handle that map from the non-linear parameters to the linear once.
- nExo : Number of exogenous variables of the model, excluding lag and constant.
- AR : true or false. true if AR specification of the MIDAS model is used.

Output:

- resid : A nobs x 1 double with the residuals of the model.

See also:

[nb_midasMapToLinear](#), [nb_midasFunc](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mlarima**

results = nb_mlarima(y,p,i,q)
results = nb_mlarima(y,p,i,q,sp,sq,constant,z,x,varargin)

Description:

Estimate a ARIMA(p,i,q) or ARIMAX(p,i,q) model using maximum likelihood.

$$\begin{aligned}y(t) &= b \cdot z + u(t) \\u(t) &= \lambda \cdot u(t-1) + c \cdot q + \epsilon(t)\end{aligned}$$

Input:

- y : A nobs x 1 double vector with the time-series.
- p : The AR degree. As a number.
- i : The degree of integration. I.e. the number of times the time-series have to be differenced.
- q : The MA degree. As a number.
- sp : Seasonal autoregressive (SAR) component. E.g. 0 to exclude, 4 for quarterly data, 12 for monthly.
- sq : Seasonal moving average (SMA) component. E.g. 0 to exclude, 4 for quarterly data, 12 for monthly.
- constant : Give 1 if a constant should be included, otherwise 0. Default is 1.
- z : Exogenous regressors in the observation equation. As a nobs x nvards. Default is [].
- x : Exogenous regressors in the transition equation. As a nobs x mvars. Default is [].

Optional inputs:

- 'stabilityTest' : true or false. If true, stability is forced on the model during estimation. Default is true.
- 'test' : true or false. If true an error will be thrown if the model is not stable. Default is true.
- 'optimizer' : See the output from the nb_getOptimizers('arima').
- 'filter' : Give true if you want to run the kalman filter to estimate the residual. Otherwise give 1. Default is false.
- 'covrepair' : Give true to repair the covariance matrix of the estimated parameters if found to not be positive definite. Default is false, i.e. to throw an error if the covariance matrix is not positive definite.
- 'options' : Inputs given to the optimset function. If no optional inputs are given default settings are used. Must be given as a struct or a cell array. If given as a cell array it is given to the optimset function. struct is recommended. See nb_getDefaultOptimset.

Output:

- results : A struct consisting of:

- > beta : The estimated coefficients. Order; constant, AR, MA, SAR, SMA then exogenous.
- > stdBeta : The standard deviation of the estimated coefficients. Order; constant, AR, MA, SAR, SMA then exogenous.
- > tStatBeta : The t-statistics of the estimated coefficients. Order; constant, AR, MA, SAR, SMA then exogenous.
- > pValBeta : The p-values of the estimated coefficients. Order; constant, AR, MA, SAR, SMA then exogenous.
- > sigma : Estimated std of the residual.
- > omega : Covariance matrix of the estimated coefficients including the residual std. Order; constant, AR, MA, SAR, SMA, exogenous then residual std.
- > likelihood : The log likelihood.
- > residual : The filter estimate of the residual. As a nsample x 1 double. Only if the filter input is set to 1.
- > X : Will be empty.
- > y : The model dependent variable. As a nsample x 1 double.
- > u : The residual from the observation equation. If no constant and z is empty this will be equal to y.
- > z : The exogenous variables included in the observation equation.
- > x : The exogenous variables included in the transition equation.

See also:

[nb_getDefaultOptimset](#), [nb_getOptimizers](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_neweyWestRobustSTD**

stdBeta = nb_neweyWestRobustSTD(x, residual)

Description:

Estimation of Newey-West heteroscedasticity and autocorrelation robust standard errors.

See page 36 of Eviews User's Guide 2.

Input:

- x : The regressors of the equation, as a nobs x nvar double.
- u : Residual from regression, as a nobs x neq double.
- xpxi : $(x'x)^{-1}$. If not given it will be calculated.
- restr : A 1 x neq where each element is a 1 x nxvar + extra logical. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included. Each logical element must be true to include in the regression equation of the corresponding equation.

Output:

- stdBeta : Std as a nvar x neq double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_nls

```
[message,beta,stdBeta,tStatBeta,pValBeta,residual,omega] = ...
    nb_nls(init,ub,lb,opt,optimizer,covrepair,func,y,x,constant, ...
    varargin)
```

Description:

Estimate multiple (unrelated) equations using nls.

$y = X \cdot func(beta) + residual$, $residual \sim N(0, residual' \cdot residual/nobs)$ (1)

Input:

- init : Initial values of the parameters of the model to estimate. As a nParam x nEq double.
- ub : Upper bound on the parameters of the model to estimate. As a nParam x nEq or nParam x 1 double. If given as empty inf is default. Only applies if optimizer is set to 'fmincon'.
- lb : Lower bound on the parameters of the model to estimate. As a nParam x nEq or nParam x 1 double. If given as empty -inf is default. Only applies if optimizer is set to 'fmincon'.

- opt : See the optimset function for more on this input. Default values will be used if it is given as [].
- optimizer : The optimizer to be used; 'fminunc', 'fminsearch' or 'fmincon'. 'fmincon' is default, i.e. if not 'fminunc' or 'fminsearch' is provided.
- covrepair : Give true to repair the covariance matrix of the estimated parameters if found to not be positive definite. Default is false, i.e. to throw an error if the covariance matrix is not positive definite.
- func : A function handle that takes the inputs in the following order:
 - > One column from the parameter matrix (i.e. the parameters for one equation at the time)
 - > One column from the y input, i.e. a double with size nobs x 1.
 - > The x input.
 - > The constant input.
 - > The rest of the optional inputs given to this function (varargin)

The output should be a nobs x 1 double with the residuals given the provided parameters.
- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.

Optional input:

- 'Aeq' : Aeq*beta = Beq (linear constraints).
- 'Beq' : Aeq*beta = Beq (linear constraints).
- 'NONLCON' : A function handle that return the values of C(beta) and Ceq(beta), where C(beta) <= 0 and Ceq(beta) == 0. E.g. [C,Ceq] = nonlinconstr(beta).
- varargin : The rest will be given as optional inputs to the func input.

Output:

- message : Non-empty if some error are thrown during estimation.
- beta : A (constant + nxvar) x neq matrix with the estimated parameters.
- stdBeta : A (constant + nxvar) x neq matrix with the standard

deviation of the estimated parameters.

- tStatBeta : A (constant + nxvar) x neq matrix with t-statistics of the estimated parameters.
- pValBeta : A (extra + nxvar) x neq matrix with the p-values of the estimated parameters.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- omega : Covariance matrix of estimated parameters. As (extra + nxvar) x (extra + nxvar) x neq

See also
nb_midasFunc

Written by Kenneth S. Paulsen

◆ nb_nwhishart

```
beta = nb_nwhishart(y,x)
[beta,sigma,residual,X] = nb_nwhishart(y,x,prior,constant,timeTrend)
```

Description:

Estimate multiple (related) equations using Normal-Wishart priors.

$y = X\beta + \text{residual}$, $\text{residual} \sim N(0, \sigma)$ (1)

Prior on β : $N(A_{prior}, V_{prior})$
Prior on σ : $IW(S_{prior}, v_{prior})$

Input:

- y : A double matrix of size T x Q of the dependent variable of the regression(s).
- x : A double matrix of size T x N of the right hand side variables of all equations of the regression.
- prior : A structure with the prior specification. If not given the default priors are:
 - > A_prior : 0
 - > V_prior : Identity matrix times 10.
 - > S_prior : 0
 - > v_prior : N + 1
 - > draws : 1For more on this prior see nb_nwhishartPrior.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false.

- timeTrend : If a linear time trend is wanted in the estimation.
Will be added first/second in the right hand side
variables. (First if constant is not given, else
second). Default is false.

Output:

- beta : A (extra + nxvar) x nEq x draws matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- sigma : A nEq x nEq x draws matrix with the covariance matrix of residuals.
- residual : Residual from the estimated equation. As an nobs x nEq matrix. If draws > 1, this is calculated at the mean.
- X : Regressors including constant and time-trend. If more than one equation this will be given as kron(I,x).

See also
`nb_nwhishartPrior`, `nb_bayesEstimator`, `nb_singleEq`

Written by Kenneth S. Paulsen

◆ **nb_nwhishartPrior**

`prior = nb_nwhishartPrior(varargin)`

Description:

Set the priors of the type:

`y = X*beta + residual, residual ~ N(0,sigma) (1)`

`y` has size `T x Q`
`X` has size `T x N`

Prior on `beta` : `N(A_prior,V_prior)`
Prior on `sigma` : `IW(S_prior,v_prior)`

- `A_prior` : A scalar double (automatically expanded) or a double with size `N x Q`
- `V_prior` : A scalar double (automatically expanded) or a double with size `(N x Q) x 1` or `(N x Q) x (N x Q)`.
- `S_prior` : A scalar double (automatically expanded) or a double with size `Q x 1`.
- `v_prior` : A scalar double.

Optionsl input:

- Run without inputs to get default priors.
- Run with 'list' as the first input to get a cellstr with the supported prior options.

- Run with 'help' to get a char with help on each option.
- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:

[nb_nwhishart](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_ols**

```
beta = nb_ols(y,x)
[beta, stdBeta, tStatBeta, pValBeta, residual, x] = ...
    nb_ols(y,x,constant,timeTrend)
```

Description:

Estimate multiple (unrelated) equations using ols.

$y = X\beta + \text{residual}$, $\text{residual} \sim N(0, \text{residual}' * \text{residual}/nobs)$ (1)

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second). Default is false.
- stdType : - 'w' : Will return White heteroskedasticity robust standard errors.
- 'nw' : Will return Newey-West heteroskedasticity and autocorrelation robust standard errors.

To estimate the covariance matrix of the estimated parameters the bartlett kernel is used with the bandwidth set to $\text{floor}(4(T/100)^{(2/9)})$ as Eviews also uses.

- 'h' : Will return homoskedasticity only robust standard errors. Default.

Output:

- beta : A (extra + nxvar) x neq matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- stdBeta : A (extra + nxvar) x neq matrix with the standard deviation of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- tStatBeta : A (extra + nxvar) x neq matrix with t-statistics of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- pValBeta : A (extra + nxvar) x neq matrix with the p-values of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- x : Regressors including constant and time-trend. If more than one equation this will be given as $\text{kron}(I, x)$.

See also
`nb_olsEstimator`, `nb_singleEq`

Written by Kenneth S. Paulsen

◆ **nb_olsRestricted**

```
beta = nb_ols(y,x)
[beta,stdBeta,tStatBeta,pValBeta,residual,x] = ...
    nb_olsRestricted(y,x,restrictions,constant,timeTrend)
```

Description:

Estimate multiple (unrelated) equations using ols.

$y = X\beta + \text{residual}$, $\text{residual} \sim N(0, \text{residual}' * \text{residual}/nobs)$ (1)

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression.
- restr : A 1 x neq cell where each element is a 1 x nxvar + extra logical. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included. Each logical element must be true to include in the regressor in the corresponding equation.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second). Default is false.
- stdType : - 'w' : Will return White heteroskedasticity robust standard errors.
- 'nw' : Will return Newey-West heteroskedasticity and autocorrelation robust standard errors.

To estimate the covariance matrix of the estimated parameters the bartlett kernel is used with the bandwidth set to floor(4(T/100)^(2/9)) as Eviews also uses.

- 'h' : Will return homoskedasticity only robust standard errors. Default.

Output:

- beta : A (extra + nxvar) x neq matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- stdBeta : A (extra + nxvar) x neq matrix with the standard deviation of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- tStatBeta : A (extra + nxvar) x neq matrix with t-statistics of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- pValBeta : A (extra + nxvar) x neq matrix with the p-values of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- residual : Residual from the estimated equation. As an nobs x neq matrix.

- x : Regressors including constant and time-trend.

See also

nb_olsEstimator, nb_singleEq

Written by Kenneth S. Paulsen

◆ nb_pca

```
F = nb_pca(X)
[F,LAMBDA,R,varF,expl,c,sigma,e] = nb_pca(X,r,method,varargin)
```

Description:

Calculate principal component using either eigenvalue decomposition or single value decomposition.

The following equation applies:

```
X = (c + F*LAMBDA + e).*sigma, e ~ N(0,R) (1)
```

Input:

- x : The data as a nobs x nvar double.

- r : The number of principal component. If empty this number will be found by the Bai and Ng (2002) test, see the optional option crit below, i.e choose the CT part of the selection criterion; $\log(V(ii,F)) + CT$. Where;

```
V(ii,F) = sum(diag(e_ii'*e_ii/T))/ii
```

and e_ii is the residual when ii factors are used

- method : Either 'svd' or 'eig'. Default is 'svd'.

> 'svd' : Uses single value deomposition to construct the principal components. This is the numerically best way, as the 'eig' method could be very unprecise!

> 'eig' : Uses the eigenvalue decomposition approach instead

- Z : Normalized (and rebalanced) version of X.

Optional inputs:

- 'crit' : You can choose between the follwing selection criterion

> 1,10: ii*(NT1/NT)*log(NT/NT1); % IC_p1 and PC_p1 (default)

> 2,11: ii*(NT1/NT)*log(CNT2); % IC_p2 and PC_p1

> 3,12: ii*log(CNT2)/CNT2; % IC_p3 and PC_p1

> 4: ii*2/T; % AIC_1

```

> 5: ii*log(T)/T; % BIC_1
> 6: CT = ii*2/N; % AIC_2
> 7: ii*log(N)/N; % BIC_2
> 8: ii*2*(NT1/NT); % AIC_3
> 9: (ii*NT1*log(NT))/NT; % BIC_3

```

where NT1 = N + T, CNT2 = min(N,T), NT = N*T and ii = 1:rMax.

- 'rMax' : The maximal number of factors to test for. Must be less than or equal to N. Default is N.
- 'trans' : Give 'demean' if you want to demean the data in the matrix X. Give 'standardize' (default) if you want to standardise the data in the matrix X. Give 'none' to drop any kind of transformation of the data in the matrix X.

Caution: To get back X when 'demean' is given you need to add the constant terms in c. I.e;

```
X = c(ones(T,1),1) + F*LAMBDA + e, e ~ N(0,R) (1*)
```

To get back X when 'standardise' you need to add the constant term and multiply with sigma. I.e.

```
X = c(ones(T,1),:) + F*LAMBDA.*sigma(ones(T,1),:)
      eps, eps ~ N(0,R*) (1*)
```

Be aware that the eps differ from e, as e is the residual from the standardised regression.

- 'unbalanced' : true or false. Set to true to allow for unbalanced dataset. Default is false. If false all rows of the dataset containing nan value are removed from calculations. The EM algorithm is essentially the one given in the paper "Macroeconomic Forecasting Using Diffusion Indexes" by Stock and Watson (2002). The algorithm is initialized by filling in missing values with the function nb_estimator.fillInForMissing.
Caution: If all variables has leading or trailing nan, these observations will be discarded.

Output:

- F : The principal component, as a nobs x r double.
- LAMBDA : The estimated loadings, as a r x nvar
- R : The covariance matrix of the residual in (1), as a nvar x nvar double.
- varF : A 1 x r double. Each column will be the variance of the corresponding column of F.
- expl : The percentage of the total variance explained by each

```

    principal component. As a 1 x r double

- c      : Constant in the factor equation. Double with 1 x nvar.

- sigma   : See 'trans'. Double with size 1 x nvar.

- e       : Residual of the equation (1) above. As a nobs x nvar double.

- z       : The normalized version of the input x.

```

Examples:

```

load halld;
[F,LAMBDA,R,varF,expl] = nb_pca(ingredients)
[F2,LAMBDA2,R2,varF2,expl2] = nb_pca(ingredients,[],'eig')

```

See also:

[nb_whiten](#), [nb_ts.pca](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_qreg**

```

beta = nb_qreg(y,x)
[beta, stdBeta, tStatBeta, pValBeta, residual, x, betaB] = ...
    nb_qreg(q,y,x,constant,timeTrend,method,draws,restr,waitbar,algo)

```

Description:

Estimate multiple (unrelated) equations using quantile regression.

$y = X\beta + \text{residual}$

Caution: Statistics package needed.

Input:

```

- q      : The quantile(s) of the regression. As a row vector of
            doubles between 0 and 1. If size(y,2) this input must be
            a scalar.

- y      : A double matrix of size nobs x neq of the dependent
            variable of the regression(s).

- x      : A double matrix of size nobs x nxvar of the right
            hand side variables of all equations of the
            regression.

- constant : If a constant is wanted in the estimation. Will be
            added first in the right hand side variables.

- timeTrend : If a linear time trend is wanted in the estimation.
            Will be added first/second in the right hand side

```

variables. (First if constant is not given, else second)

- method : A string with the method to be used to bootstrap the standard errors. See nb_bootstrap for the supported methods or use 'sparsity'. 'sparsity' is default and it uses the asymptotic formulas found in; <https://support.sas.com/documentation/onlinedoc/stat/141/qreg.pdf>
- draws : Number of draws when bootstrapping the standard errors.
- restr : A 1 x neq cell where each element is a 1 x nxvar + extra logical. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included. Each logical element must be true to include in the regression of the corresponding equation. May also be empty, which is default, i.e. no restrictions added.
- waitbar : Add waitbar during estimation. Default is false. true or false.

Output:

- beta : A (extra + nxvar) x neq x numQuantiles matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- stdBeta : A (extra + nxvar) x neq matrix with the standard deviation of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- tStatBeta : A (extra + nxvar) x neq matrix with t-statistics of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- pValBeta : A (extra + nxvar) x neq matrix with the p-values of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- x : Regressors including constant and time-trend.
- betaB : Distribution of parameters produced by bootstrapping. A (extra + nxvar) x neq x numQuantiles x nDraws matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.

See also
[nb_quantileEstimator](#), [nb_singleEq](#), [nb_bootstrap](#), [linprog](#)

Written by Kenneth S. Paulsen

◆ nb_ridge

```
[beta,residual,X] = nb_ridge(y,X,k,constant)
```

Description:

RIDGE estimation. Solves the constrained optimization problem:

$$\min (y - X\beta)^2 + k \sum (\beta(i))^2$$

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- X : A double matrix of size nobs x nxvar of the right hand side variables of all equations of the regression.
- k : The value of k in the minimization problem. Default is $nxvar * \sigma_{ols}^2 / (\beta_{ols}' * \beta_{ols})$, where σ_{ols} is the ols estimator of the residual variance, and β_{ols} is the ols estimate of the model parameters. Must be a scalar number greater or equal to 0.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables. Default is false. The constant term is first estimate by the mean of y, and then y is demeaned before doing the RIDGE estimation.

Output:

- beta : A (extra + nxvar) x neq matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- X : Regressors including constant. If more than one equation this will be given as $kron(I, x)$.

See also:

[nb_ols](#), [nb_lasso](#)

Written by Kenneth Åkerblom Paulsen

◆ nb_setUpForDiffuseFilter

```
[M,G,C,x0,PS0,PINFO,fail] = nb_setUpForDiffuseFilter(H,A,B)
```

Description:

Set up for diffuse Kalman filtering.

See also:

[nb_kalmanSmootherDiffuseDSGE](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_tsls

```
beta = nb_tsls(y,x,z,inst)
[beta, stdBeta, tStatBeta, pValBeta, residual, x] = ...
    nb_tsls(y,x,z,inst,constant,timeTrend)
```

Description:

Estimate multiple (unrelated) equations using tsls.

Input:

- y : A double matrix of size nobs x neq of the dependent variable of the regression(s).
- x : A double matrix of size nobs x nxvar of the exogenous variables of all equations of the regression.
- z : A double matrix of size nobs x nzvar of the endogenous variables of all equations of the regression.
- inst : A 1 x nzvar cell of the instruments for each endogenous variable. Each element must be a double with size nobs x nzivar. nzivar must be greater than 0.
- constant : If a constant is wanted in the estimation. Will be added first in the right hand side variables.
- timeTrend : If a linear time trend is wanted in the estimation. Will be added first/second in the right hand side variables. (First if constant is not given, else second)

Output:

- beta : A (extra + nxvar + nzvar) x neq matrix with the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- stdBeta : A (extra + nxvar + nzvar) x neq matrix with the standard deviation of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant

and/or time trend is included.

- tStatBeta : A (extra + nxvar + nzvar) x neq matrix with t-statistics of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- pValBeta : A (extra + nxvar + nzvar) x neq matrix with the p-values of the estimated parameters. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included.
- residual : Residual from the estimated equation. As an nobs x neq matrix.
- x : Regressors of final regression including constant, time-trend and instrumented endogenous variables.

See also

`nb_tslsEstimator`, `nb_singleEq`

Written by Kenneth S. Paulsen

◆ **`nb_varStateSpace`**

```
[d,H,R,c,A,B,Q,G,obs] = nb_arimaStateSpace(par,nExo)
```

Description:

Returns a state-space representation of an VARX(nLags) model.

Observation equation:

$y = d + Hx + v$

State equation:

$x = c + Ax_{-1} + Gz + Bu$

Input:

- par : The parameter vector of the VARX model.
 - Order:
 - constant
 - exogenous
 - lagged endogenous
 - residual std
- nDep : The number of dependent variables of the model.
- nLags : the number of lags of the VAR.
- constant : Include constant or not. true or false.
- nExo : Number of exogenous variables included in the model.
- restr : A nDep x nDep*nLags logical matrix. true fro the

non-restricted parameters.

- restrVal : Values of the restricted parameters.

Output:

- See equation above

- obs : Index of observables in the state vector.

Examples:

See also:

[nb_kalmanlikelihood](#)

Written by Kenneth Åtterhagen Paulsen

◆ **nb_whiteRobustSTD**

stdBeta = nb_whiteRobustSTD(x, residual)

Description:

Estimation of White heteroscedasticity robust standard errors.

See page 35 of Eviews User's Guide 2.

Input:

- x : The regressors of the equation, as a nobs x nvar double.
- u : Residual from regression, as a nobs x neq double.
- xpxi : $(x'x)^{-1}$. If not given it will be calculated.
- restr : A 1 x neq where each element is a 1 x nxvar + extra logical. Where extra is 0, 1 or 2 dependent on the constant and/or time trend is included. Each logical element must be true to include in the regression equation of the corresponding equation.

Output:

- stdBeta : Std as a nvar x neq double.

Written by Kenneth Åtterhagen Paulsen

◆ **nb_whiten**

```
F = nb_whiten(X,r)
F = nb_whiten(X,r,varargin)
```

Description:

Calculate of factors from X that have variance one and covariances 0.

Input:

- X : The data as a nobs x nvar double.
- r : The number of principal component. If empty nvar factors are returned.

Optional input:

- 'missing' : Either '' (don't handle missing observations), 'zeros' or 'interpolate'.

Output:

- F : The principal component, as a nobs x r double.

Examples:

```
load hald;
F = nb_whiten(ingredients)
```

See also:

[nb_pca](#), [nb_ts.whiten](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_zeroSpectrumEstimation**

```
[lambda,gamma,bandWidth] = nb_zeroSpectrumEstimation(residual, ...
freqZeroSpectrumEstimator,bandWidth,bandWithCrit)
```

Description:

Estimation of frequency zero spectrum.

Input:

- u : Residual from regression nobs x 1 double.
- kernel : Either:
 - > 'bartlett' : Bartlett kernel function.
Default.

```
> 'parzen'      : Parzen kernel function.  
> 'quadratic'  : Quadratic Spectral kernel  
function.  
  
- bandWidth     : The selected band width of the frequency zero  
spectrum estimation. Default is 3.  
  
- bandWithCrit : Band width selection criterion. Either:  
  
    > 'nw'       : Newey-West selection method.  
                    Default.  
  
    > 'a'        : Andrews selection method. AR(1)  
                    specification.  
  
    > ''         : Manually set by the bandWidth  
                    input.
```

Output:

- lambda : Freq zero spectrum. As a double.
- gamma : Autocorrelation. As a 1 x bandWidth double.
- bandWidth : Selected band width.

Written by Kenneth Sæterhagen Paulsen

25.17 Forecasting

- nb_calculatePIT
- nb_computeForecastAnticipated
- nb_computeMidasForecast
- nb_defaultWeights
- nb_estimateKernelDensity
- nb_evaluateDensityForecast
- nb_evaluateForecast
- nb_forecast.addZContribution
- nb_forecast.buildShockRestrictions
- nb_forecast.checkSVDPrior
- nb_forecast.computeBoundedForecasts
- nb_forecast.conditionalOnDistributionEngine
- nb_forecast.createParameterUncertaintyWaitbar
- nb_forecast.createWaitbar
- nb_forecast.densityBootstrap
- nb_forecast.densityBootstrapStepAheadFactorModel
- nb_forecast.densityForecast
- nb_forecast.densityPosterior
- nb_forecast.drawMoreFromPosterior
- nb_forecast.drawNowcastFromKalmanFilter
- nb_forecast.drawShocksAndExogenous
- nb_forecast.evaluateDensityForecast
- nb_forecast.evaluatePointForecastLowFreq
- nb_forecast.exprModel
- nb_forecast.exprModelForecast
- nb_forecast.fillInForMissing
- nb_forecast.getActual
- nb_computeForecast
- nb_computeForecastAnticipatedBP
- nb_conditionalSecondOrderMoments
- nb_dieboldMarianoTest
- nb_evaluateDensity
- nb_evaluateDensityForecastOnlyScore
- nb_forecast
- nb_forecast.boundedConditionalProjectionEngine
- nb_forecast.checkForMissing
- nb_forecast.cleanUpInputs
- nb_forecast.condShockForecastEngine
- nb_forecast.conditionalProjectionEngine
- nb_forecast.createReportedVariables
- nb_forecast.defaultInputs
- nb_forecast.densityBootstrapFactorModel
- nb_forecast.densityBootstrapStepAheadModel
- nb_forecast.densityFoundReplic
- nb_forecast.doMeasurementEq
- nb_forecast.drawNowcast
- nb_forecast.drawRegimes
- nb_forecast.estimateAndBootstrapX
- nb_forecast.evaluatePointForecast
- nb_forecast.expandRestrictionsForBoundedForecast
- nb_forecast.exprModelDensityForecast
- nb_forecast.exprModelPointForecast
- nb_forecast.forecastStochasticTrend
- nb_forecast.getDeterministic

- nb_forecast.getDeterministicVariables
- nb_forecast.getForecastVariables
- nb_forecast.getModelMatrices
- nb_forecast.getSVProcess
- nb_forecast.getStartingValues
- nb_forecast.interpretBins
- nb_forecast.isMissingVar
- nb_forecast.map2Density
- nb_forecast.midas
- nb_forecast.midasPointForecast
- nb_forecast.multvnrnd
- nb_forecast.prepareRestrictions
- nb_forecast.removeBeforeEvaluated
- nb_forecast.setInitialValues2CondDB
- nb_forecast.tvp
- nb_forecast.undiffARIMA
- nb_hdi
- nb_irfEngine
- nb_irfEngine.collect
- nb_irfEngine.doParallel
- nb_irfEngine.irfFactorModelBootstrap
- nb_irfEngine.irfPoint
- nb_irfEngine.irfPosterior
- nb_irfEngine.normalizeToMean
- nb_kernelCDFBounds
- nb_makeArtificial
- nb_restrictedWeights
- nb_simulateFromDensity
- nb_varDecomp
- nb_forecast.getEvaluatedVariables
- nb_forecast.getMissingFromCondInfo
- nb_forecast.getQuantileModel
- nb_forecast.getStartAndEnd
- nb_forecast.getStateSpace
- nb_forecast.interpretRestrictions
- nb_forecast.kalmanConditionalProjectionEngine
- nb_forecast.mapToObservablesMFVAR
- nb_forecast.midasDensityForecast
- nb_forecast.midasPrepareForReporting
- nb_forecast.pointForecast
- nb_forecast.quantileDensityForecast
- nb_forecast.saveToFile
- nb_forecast.setUpForBoundedForecast
- nb_forecast.uncondForecastEngine
- nb_getDensityPoints
- nb_identifyShocks
- nb_irfEngine.checkInputFile
- nb_irfEngine.createReportedVariables
- nb_irfEngine.irfBootstrap
- nb_irfEngine.irfIdentDraws
- nb_irfEngine.irfPointStochTrend
- nb_irfEngine.irfWithStochasticTrend
- nb_irfEngine.report
- nb_kolsrudJointPredictionBands
- nb_mincerZarnowitz
- nb_scoreType2evalType

► **nb_forecast.addZContribution ↑**

```
[Y,Z] = nb_forecast.addZContribution(Y,model,options,restrictions, ...
inputs,draws,iter)
```

Description:

Add contributions of exogenous variables to idiosyncratic component of the observation equation.

X = G*Z + Y

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.boundedConditionalProjectionEngine ↑**

```
[Emean,Xmean,states,solution]...
= nb_forecast.boundedConditionalProjectionEngine(Y0,A, ...
B,CE,ss,Qfunc,nSteps,restrictions,solution,inputs)
```

Description:

Identify the shocks/residuals to meet the conditional restriction given bounds on some endogenous variables.

See also:

[nb_forecast.conditionalProjectionEngine](#)
[nb_forecast.condShockForecastEngine](#)
[nb_forecast.pointForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.buildShockRestrictions ↑**

```
[R,shocksNumOfAnt,states] = nb_forecast.buildShockRestrictions(AA,B, ...
endoRestInd,numRestShocks,numCondPer,numAntPer,states,append)
```

Description:

Build the a matrix representing the restrictions made by the conditional information. Used to identify the shocks/residual to match the conditional information.

See Kenneth Sæterhagen Paulsen (2010), "Conditional forecast in DSGE models - A conditional copula approach".

Implements the steps in section 3 of setting up the matrix R.

The model is on the form $y(t) = A y(t-1) + \sum_{j=1}^n [B(j) \epsilon_j(t+j)]$

Inputs:

- AA : Model transition matrix raised to the $j - 1$, i.e. page AA(1) = A, AA(2) = A^2 and so on. As a nEndo x nEndo x j double.

- B : Impact matrices for the shocks. (Also anticipated). As a nEndo x nExo x nAnt double.

Outputs:

- R : See equation (20) on page 7 of Kenneth Sæterhagen Paulsen (2010).

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.checkForMissing ↑**

```
[nowcast,missing] = nb_forecast.checkForMissing(options,inputs,dep)
```

Description:

Check for missing data at the end of sample. Only handle one missing observation at the end here!

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.checkSVDPrior ↑**

```
svd = nb_forecast.checkSVDPrior(options)
```

Description:

Check if the stochastic-volatility-dummy prior is used for a VAR model.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.cleanUpInputs ↑**

```
inputs = nb_forecast.cleanUpInputs(inputs)
```

Description:

Clean up inputs so it may be given back to nb_model_generic.forecast when object is being updated.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.computeBoundedForecasts ↑**

```
[Y,E] = computeBoundedForecasts(A,B,C,ss,YF,restrictions,MUX,MUs,inputs,...  
solution,lower,upper,belowL,aboveL,Y,E,iter)
```

Description:

Compute bounded forecast using expected shocks.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.condShockForecastEngine** ↑

```
[Y,states,PAI] = nb_forecast.condShockForecastEngine(y0,A,B,C,ss,Qfunc,MUX,...  
MUs,states,restrictions,nSteps,inputs)
```

Description:

Produce forecast conditional on shocks only forecast with or without
Markov-switching

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.conditionalOnDistributionEngine** ↑

```
[E,X,states,solution] = nb_forecast.conditionalOnDistributionEngine(y0,  
model,ss,nSteps,draws,restrictions,...  
solution)
```

Description:

Conditional on restrictions given by marginal distributions. A copula is used to draw from the marginal given a correlation matrix. For each draw the shocks/residual to match the drawn conditional information is identified. And as a result you end up with the wanted number of draws from the distributions of the shocks/residual to match the conditional information.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.conditionalProjectionEngine** ↑

```
[Emean,Xmean,states,solution]...  
= nb_forecast.conditionalProjectionEngine(Y0,model,ss,nSteps,...  
restrictions,solution,s)
```

Description:

Identify the shocks/residuals to meet the conditional restriction.

See Kenneth Sæterhagen Paulsen (2010), "Conditional forecast in DSGE models - A conditional copula approach".

Implements the steps in section 3.1.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.createParameterUncertaintyWaitbar** ↑

```
[h,inputs] = nb_forecast.createParameterUncertaintyWaitbar(inputs)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.createReportedVariables** ↑

```
[Yout,dep] = nb_forecast.createReportedVariables(options,inputs,Y,...  
dep,start,iter)
```

Description:

Create reported variables based on simulated forecast.

See also:

[nb_forecast.pointForecast](#), [nb_forecast.densityForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.createWaitbar** ↑

```
[h,iter,closeWaitbar] = nb_forecast.createWaitbar(inputs,start,finish)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.defaultInputs** ↑

```
default = nb_forecast.defaultInputs(convert)
```

Description:

Get default struct to give as the input to the inputs input of the nb_forecast function.

Input:

- convert : Convert to struct if set to true, otherwise not. Default is true.

Output:

- default : A N x 3 cell matrix or a struct with N fields.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityBootstrap ↑**

```
[Y,XE,solution] = nb_forecast.densityBootstrap(y0,restrictions, ...
                                                 model,options,results,nSteps,iter,inputs,solution)
```

Description:

Produce density forecast of a model using bootstrap.

This function does not handle Markov switching models!

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityBootstrapFactorModel ↑**

```
[Y,XE,solution] = nb_forecast.densityBootstrapFactorModel(y0, ...
                                                 restrictions,model,options,results,nSteps,iter,inputs,solution)
```

Description:

Produce density forecast of a factor model using bootstrap.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityBootstrapStepAheadFactorModel ↑**

```
[Y,XE] = nb_forecast.densityBootstrapStepAheadFactorModel(y0, ...
                                                 restrictions,model,options,results,iter,inputs)
```

Description:

Produce density forecast of a step ahead factor model using bootstrap.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityBootstrapStepAheadModel** ↑

```
[Y,XE] = nb_forecast.densityBootstrapStepAheadModel(...  
    restrictions,model,options,results,nSteps,iter,inputs)
```

Description:

Produce density forecast of a step ahead model using bootstrap.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityForecast** ↑

```
[fData,evalFcst,solution] = nb_forecast.densityForecast(y0,...  
    restrictions,model,options,results,nSteps,start,iter,actual,...  
    inputs,solution)
```

Description:

Produce density forecast with wanted method

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityFoundReplic** ↑

```
[Y,XE,states,PAI] = nb_forecast.densityFoundReplic(y0,...  
    restrictions,model,options,results,nSteps,iter,inputs)
```

Description:

Produce density forecast using already solved models.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.densityPosterior** ↑

```
[Y,XE,states,solution] = nb_forecast.densityPosterior(y0,restrictions,...  
    model,options,results,nSteps,draws,parameterDraws,iter,inputs,...  
    solution)
```

Description:

Produce density forecast using posterior draws.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.doMeasurementEq** ↑

```
[dep,Y] = nb_forecast.doMeasurementEq(options,results,inputs,nSteps,...  
model,Y,X,dep,iter)
```

Description:

Produce forecast of measurement equation of a model with an observation equation.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.drawMoreFromPosterior** ↑

```
[extra,betaD,sigmaD,yD,pD] = nb_forecast.drawMoreFromPosterior(inputs,...  
coeffDraws)
```

Description:

Check if there has been done enough posterior draws. If there isn't we draw more.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.drawNowcast** ↑

```
y0 = nb_forecast.drawNowcast(options,results,model,inputs,iter)
```

Description:

Produce density nowcast.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.drawNowcastFromKalmanFilter** ↑

```
y0 = nb_forecast.drawNowcastFromKalmanFilter(yD,pD,draws)
```

Description:

Produce density nowcast from kalman filter output.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.drawRegimes ↑**

```
[E,X,states,solution] = nb_forecast.drawRegimes(y0,model,ss,...  
nSteps,draws,restrictions,solution,inputs,options)
```

Description:

Draw regimes.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.drawShocksAndExogenous ↑**

```
[E,X,solution] = nb_forecast.drawShocksAndExogenous(y0,model,ss,nSteps,  
draws,restrictions,solution,inputs,options)
```

Description:

Draw from the shock distribution given the conditional information.

For Markov switching model this method return only one simulated horizon of states. I.e. to take into account the state uncertainty you need to loop this function. This give much more flexibility for the user on how to produce the density forecast.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.estimateAndBootstrapX ↑**

```
XAR = nb_forecast.estimateAndBootstrapX(opt,restr,draws,index,...  
inputs,type)
```

Description:

This is the function that project the exogenous variables of the model when the 'exoProj' input is not empty.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.evaluateDensityForecast](#)** ↑

```
[fData,evalFcst] = nb_forecast.evaluateDensityForecast(Y,actual,dep,...  
model,options,inputs)
```

Description:

Evalaute density forecasts.

See also:

[nb_forecast.densityForecast](#), [nb_forecast.exprModelDensityForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.evaluatePointForecast](#)** ↑

```
evalFcst = nb_forecast.evaluatePointForecast(Y,actual,dep,model,inputs)
```

Description:

Evaluate point forecast.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.evaluatePointForecastLowFreq](#)** ↑

```
evalFcst = nb_forecast.evaluatePointForecastLowFreq(evalFcst,options,...  
Y,dep,model,inputs,start)
```

Description:

Evaluate point forecast of low frequency variable of mixed frequency models.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.expandRestrictionsForBoundedForecast](#)** ↑

```
restrictions = nb_forecast.expandRestrictionsForBoundedForecast(...  
restrictions,nSteps,model)
```

See also:

`nb_forecast.pointForecast`

Written by Kenneth Sæterhagen Paulsen

► `nb_forecast.exprModel` ↑

```
fcst = nb_forecast.exprModel(model,options,results,startInd,endInd,...  
nSteps,inputs)
```

Description:

Produce forecast of a model using expressions, see the `nb_exprModel` class.

Written by Kenneth Sæterhagen Paulsen

► `nb_forecast.exprModelDensityForecast` ↑

```
[fData,evalFcst] = nb_forecast.exprModelDensityForecast(Z,...  
restrictions,model,options,results,nSteps,iter,actual,inputs)
```

Description:

Produce density forecast of a model using expressions using bootstrap.

This function does not handle Markov switching models!

Written by Kenneth Sæterhagen Paulsen

► `nb_forecast.exprModelForecast` ↑

```
[Y,X,dep] = nb_forecast.exprModelForecast(Z,E,nSteps,options,...  
restrictions,results,inputs,model,iter)
```

Description:

Core function for producing forecast from a model using expressions.

This function does not handle Markov switching models!

See also:

[nb_forecast.exprModelPointForecast](#), [nb_forecast.exprModelDensityForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.exprModelPointForecast](#)** ↑

```
[Y,evalFcst] = nb_forecast.exprModelPointForecast(Z,restrictions,...  
model,options,results,nSteps,iter,actual,inputs)
```

Description:

Produce point forecast of a model using expressions.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.fillInForMissing](#)** ↑

```
MUc = nb_forecast.fillInForMissing(DYbar,myv,nVarY,nCondPer,...  
mev,nVarE,nCondPerE)
```

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.forecastStochasticTrend](#)** ↑

```
[Y,X,E] = nb_forecast.forecastStochasticTrend(y0,restrictions,solution,...  
options,results,nSteps,iter,inputs)
```

Description:

Produce forecast of a DSGE model with a stochastic trends.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.getActual](#)** ↑

```
actual = nb_forecast.getActual(options,inputs,model,nSteps,dep,startFcst,split)
```

Description:

Get historical data to compare forecast with.

See also:

[nb_forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.getDeterministic`** ↑

```
[X,exo] = nb_forecast.getDeterministic(options,inputs,nSteps,exo,constant)
```

Description:

Add deterministic exogenous vars to conditional information.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.getDeterministicVariables`** ↑

```
deterministic = nb_forecast.getDeterministicVariables(exo)
```

Description:

Get the deterministic exogenous variables.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.getEvaluatedVariables`** ↑

```
[Y,dep] = nb_forecast.getEvaluatedVariables(Y,actual,dep,model,inputs)
```

Description:

Get evaluated variables and forecast.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.getForecastVariables`** ↑

```
dep = nb_forecast.getForecastVariables(options,model,inputs,type)
```

Description:

Get forecast variables from model

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.getMissingFromCondInfo`** ↑

```
[nowcast,missing] = nb_forecast.getMissingFromCondInfo(...  
    options,inputs,restrictions)
```

Description:

Get missing info when conditional information is used. I.e. the conditional information introduce ragged-edge in forecast.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_forecast.getModelMatrices** ↑

```
model = nb_forecast.getModelMatrices(model,iter,irf,options,nSteps)
```

Description:

Get the model matrices of a given iterative estimation.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_forecast.getQuantileModel** ↑

```
model = nb_forecast.getQuantileModel(options,model,quantile)
```

Description:

Get the median model when it is estimated with quantile regression.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_forecast.getSVProcess** ↑

```
s = nb_forecast.getSVProcess(options,start,nSteps,fcstOnly)
```

Description:

Get the stochastic volatility process when the stochastic-volatility-dummy prior

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_forecast.getStartAndEnd** ↑

```
[start,finish,start_est,startFcst] = nb_forecast.getStartAndEnd(...  
inputs,model,options,start,finish)
```

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.getStartingValues ↑**

```
Y0 = nb_forecast.getStartingValues(startingValues,options,solution,...  
results,last)
```

Description:

Get the selected starting values.

See also:

[nb_forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.getStateSpace ↑**

```
[x0,P0,d,H,R,T,c,A,B,Q,G] = nb_forecast.getStateSpace(model,nObs,...  
yHist,nSteps)
```

Description:

Go from the state space representation in the model struct to the state space matrices needed by the [nb_kalmansmooth_missing](#) function.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.interpretBins ↑**

```
inputs = interpretBins(options,inputs)
```

Description:

Interpret the 'bins' input.

See also:

[nb_forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.interpretRestrictions** ↑

```
[bounds, indY] = nb_forecast.interpretRestrictions(bounds, endo, shocks)
```

Description:

Interpret the restrictions given on bounded variables, e.g. zero lower bound

See also:

[nb_forecast.setUpForBoundedForecast](#)
[nb_forecast.boundedConditionalProjectionEngine](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.isMissingVar** ↑

```
missingVar = nb_mlEstimator.isMissingVar(options)
```

Description:

Is the model a missing observation VAR or MF-VAR model estimated with using a kalman filter?

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.kalmanConditionalProjectionEngine** ↑

```
[E,Xmean,solution] = nb_forecast.kalmanConditionalProjectionEngine(...  
    draws,model,s,nSteps,restrictions,solution)
```

Description:

Uses the kalman filter to draw the shocks distributions to match the uncertainty of the endogenous variables returned by the kalman filter.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.map2Density** ↑

```
[Y,X] = nb_forecast.map2Density(dist,Y,X)
```

Description:

Map normalized data to estimated distribution.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_forecast.mapToObservablesMFVAR** ↑

```
[Y,dep] = nb_forecast.mapToObservablesMFVAR(Y,options,model,inputs,start)
```

Description:

Get forecast on the observable variables of a MF-VAR and merge with rest.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_forecast.midas** ↑

```
fcst = nb_forecast.midas(model,options,results,startInd,endInd,...  
nSteps,inputs)
```

Description:

Produce forecast of MIDAS models.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_forecast.midasDensityForecast** ↑

```
[fData,evalFcst] = nb_forecast.midasDensityForecast(y0,restrictions,...  
model,options,results,nSteps,iter,actual,inputs)
```

Description:

Produce density forecast of MIDAS models.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_forecast.midasPointForecast** ↑

```
[Y,evalFcst] = nb_forecast.midasPointForecast(y0,restrictions,model,...  
options,results,nSteps,index,actual,inputs)
```

Description:

Produce point forecast of MIDAS models.

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.midasPrepareForReporting** ↑

```
optLow = nb_forecast.midasPrepareForReporting(options)
```

Description:

Create reported variables based on simulated forecast.

See also:

[nb_forecast.midasPointForecast](#), [nb_forecast.midasDensityForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.multvnrnd** ↑

```
E = nb_forecast.multvnrnd(sigma,nSteps,nSim)
```

Description:

Draw from the multivariate normal distribution

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.pointForecast** ↑

```
[Y,evalFcst,solution] = nb_forecast.pointForecast(y0,restrictions,...  
model,options,results,nSteps,start,iter,...  
actual,inputs,solution)
```

Description:

Produce point forecast.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.prepareRestrictions`** ↑

```
restrictions = nb_forecast.prepareRestrictions(model,...  
options,results,nSteps,condDB,condDBVars,inputs,shockProps)
```

Description:

Prepare the conditional information for forecasting routines

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.quantileDensityForecast`** ↑

```
YQ = nb_forecast.quantileDensityForecast(y0,restrictions,model,...  
options,results,nSteps,iter,inputs)
```

Description:

Produce density forecast of a model that is estimated with quantile regression. I.e. one "point" forecast given one quantile.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.removeBeforeEvaluated`** ↑

```
remove = nb_forecast.removeBeforeEvaluated(model,inputs)
```

Description:

Removed variables from evaluated forecast.

Written by Kenneth Sæterhagen Paulsen

► **`nb_forecast.saveToFile`** ↑

```
[evalFcst,saveToFile] = nb_forecast.saveToFile(evalFcst,inputs)
```

Description:

Save forecast evaluation to file.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.setInitialValues2CondDB](#)** ↑

```
Y0 = nb_forecast.setInitialValues2CondDB(Y0,model,restrictions)
```

Description:

Set intial values to conditional information.

See also:

[nb_forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.setUpForBoundedForecast](#)** ↑

```
inputs = nb_forecast.setUpForBoundedForecast(nSteps,model,restrictions,inputs,nSim)
```

Description:

Prepare stuff for bounded forecast.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.tvp](#)** ↑

```
[Y,XE] = nb_forecast.tvp(restrictions,model,options,results,nSteps,...  
iter,inputs,type)
```

Description:

Produce forecast of time-varying parameter models.

Written by Kenneth Sæterhagen Paulsen

► **[nb_forecast.uncondForecastEngine](#)** ↑

```
[Y,states,AA] = nb_forecast.uncondForecastEngine(y0,A,B,ss,Qfunc,E,...  
states,transProbInit,maxiter)
```

Description:

Produce unconditional forecast with or without Markov-switching or a break point model.

See also:

[nb_forecast.conditionalProjectionEngine](#), [ms.uncondForecastEngine](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_forecast.undiffARIMA** ↑

```
Y = nb_forecast.undiffARIMA(options,start,Y)
```

Description:

Undiff the forecast to the original level of integration.

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.checkInputFile** ↑

```
[paused, canceling] = nb_irfEngine.checkInputFile(filename)
```

Description:

Check input file

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.collect** ↑

```
irfData = nb_irfEngine.collect(options,inputs,Y,dep,results,ss)
```

Description:

Collect IRFs.

See also:

[nb_irfEngine.irfPoint](#), [nb_irfEngine.irfPointStochTrend](#)

Written by Kenneth Sæterhagen Paulsen

► **`nb_irfEngine.createReportedVariables`** ↑

```
[Y,dep] = nb_irfEngine.createReportedVariables(options,inputs,y0,Y,...  
    dep,results,ss)
```

Description:

Create reported variables based on simulated forecast.

See also:

[nb_irfEngine.collect](#)

Written by Kenneth Sæterhagen Paulsen

► **`nb_irfEngine.doParallel`** ↑

```
[irfDataD,pause] = nb_irfEngine.doParallel(method,model,options,...  
    results,inputs)
```

Description:

Produce IRFs with error bands in parallel.

Written by Kenneth Sæterhagen Paulsen

► **`nb_irfEngine.irfBootstrap`** ↑

```
[irfDataD,pause] = nb_irfEngine.irfBootstrap(model,options,results,...  
    inputs)
```

Description:

Produce IRFs with error bands using bootstrap methods

Written by Kenneth Sæterhagen Paulsen

► **`nb_irfEngine.irfFactorModelBootstrap`** ↑

```
[irfDataD,pause] = nb_irfEngine.irfFactorModelBootstrap(model,...  
    options,results,inputs)
```

Description:

Produce IRFs with error bands of factor models using bootstrap methods.

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.irfIdentDraws** ↑

```
irfDataD = nb_irfEngine.irfIdentDraws(model,options,results,inputs)
```

Description:

Produce IRFs with error bands using uncertainty in identification.

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.irfPoint** ↑

```
irfData = nb_irfEngine.irfPoint(model,options,inputs,results)
```

Description:

Produce point IRF.

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.irfPointStochTrend** ↑

```
irfData = nb_irfEngine.irfPointStochTrend(modelObj,options,inputs)
```

Description:

Produce point IRF of nb_dsgen model with stochastic trend.

Written by Kenneth Sæterhagen Paulsen

► **nb_irfEngine.irfPosterior** ↑

```
[irfDataD,pause] = nb_irfEngine.irfPosterior(model,options,inputs)
```

Description:

Produce IRFs with error bands using posterior draws.

Written by Kenneth Sæterhagen Paulsen

► **[nb_irfEngine.irfWithStochasticTrend](#)** ↑

```
irfData = nb_irfEngine.irfWithStochasticTrend(solution,options,...  
      inputs,results)
```

Description:

Produce point IRF of a DSGE model with stochastic trend.

Written by Kenneth Sæterhagen Paulsen

► **[nb_irfEngine.normalizeToMean](#)** ↑

```
irfData = nb_irfEngine.normalizeToMean(options,inputs,irfData)
```

Description:

Normalize IRFs to mean.

See also:

[nb_irfEngine](#)

Written by Kenneth Sæterhagen Paulsen

► **[nb_irfEngine.report](#)** ↑

```
paused = nb_irfEngine.report(inputs,h,index,kk,jj,display)
```

Description:

Report status

Written by Kenneth Sæterhagen Paulsen

◆ **[nb_calculatePIT](#)**

```
pit = nb_calculatePIT(density, domain, actual)  
pit = nb_calculatePIT(density, domain, actual, fcst)
```

Description:

Calculate PIT of a density forecast given actual data. If the density forecasts are well calibrated, the returned output should be distributed uniformly.

Input:

- density : A nHor x nDomain double with the forecast density at forecast horizons 1 to nHor.

Or a 1 x nHor cell, where each element is a 1 x 2 cell array on the format {typeOfDistribution,parameters}. E.g. {'normal',{0,1}}. See the nb_distribution class for more. the properties type and parameters.
- domain : A nHor x nDomain double storing the domain of the density forecast. I.e.

Caution : Can also be of size 1 x nDomain, but then it is assumed that the domain is the same at all horizons!
- Caution : It is assumed that the domain is equally spaced.
- actual : A nHor x 1 double with the actual data for the horizon of interest
- fcst : If the density is taken from another forecast we need the difference in the mean forecast to move the domain accordingly. A nHor x 1 double.

Output:

- pit : A nHor x 1 double with the PITs

See also:

[nb_model_generic.getPIT](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_computeForecast

`Y = nb_computeForecast(A,B,C,Y,X,E)`

Description:

Compute forcast of equations on the form;

$$Y(:,t+1) = A*Y(:,t) + B*X + C*E$$

Input:

- A : See the equation above. A nEndo x nEndo double.
- B : See the equation above. A nEndo x nExo double.
- C : See the equation above. A nEndo x nRes double.

- Y : A nEndo x nSteps + 1 double, where the Y(:,1) element must be the initial values.
- X : A nExo x nSteps double with the exogenous forecast data.
- E : A nRes x nSteps double with the residual data. E.g. simulated shocks.

Output:

- Y : The forecast of the endogenous variables as a nendo x nSteps double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_computeForecastAnticipated

Y = nb_computeForecastAnticipated(A,B,C,Y,X,E)

Description:

Compute forcast of equations on the form;

$$Y(:,t+1) = A*Y(:,t) + B*X + C*E$$

Input:

- A : See the equation above. A nEndo x nEndo double.
- B : See the equation above. A nEndo x nExo double.
- C : See the equation above. A nEndo x nResidual double. If C have more pages, it means anticipated shocks.
- Y : A nendo x nSteps + 1 double, where the Y(:,1) element must be the initial values.
- X : A nexo x nSteps double with the exogenous forecast data.
- E : A nres x nSteps double with the residual data. E.g. simulated shocks.

Output:

- Y : The forecast of the endogenous variables as a nendo x nSteps double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_computeForecastAnticipatedBP

```
Y = nb_computeForecastAnticipatedBP(A,B,C,ss,Y,X,E,states)
```

Description:

Compute forecast of equations on the form;

```
Y(:,t+1) = ss{s(t+1)} + A{s(t+1)}*(Y(:,t) - ss{s(t+1)}) +  
B{s(t+1})*X(:,t+1) + C{s(t+1})*E(:,t+1)
```

With unanticipated switching of parameters (also possibly affecting the steady-state)

Input:

- A : See the equation above. A cell array, each element must be a nEndo x nEndo double.
- B : See the equation above. A cell array, each element must be a nEndo x nExo double.
- C : See the equation above. A cell array, each element must be a nEndo x nResidual double. If C have more pages, it means anticipated shocks.
- ss : A cell array with the steady-state of the model. Each element must be a nEndo x 1 double.
- Y : A nEndo x nSteps + 1 double, where the Y(:,1) element must be the initial values.
- X : A nExo x nSteps double with the exogenous forecast data.
- E : A nRes x nSteps double with the residual data. E.g. simulated shocks.
- states : A vector indicating which state we are going to be in at each forecasting steps. A nSteps double, not including the state of the period of the initial condition.

Output:

- Y : The forecast of the endogenous variables as a nEndo x nSteps double.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_computeMidasForecast**

```
Y = nb_computeMidasForecast(A,B,Y,X,E)
```

Description:

Compute forecast of a MIDAS model

Input:

- A : See the equation above. A 1 x 1 double or empty. If not empty a AR component is added to the MIDAS model.
- B : See the equation above. A nSteps x nexo double.
- Y : A 1 x nSteps + 1 double, where the Y(:,1) element must be the initial values.
- X : A nexo x 1 double with the exogenous variables of the model or a nexo x 1 x nSteps double (if the regressors change with the forecasting step (e.g. MIDAS beta lag model!))
- E : A 1 x nSteps double with the residual data. E.g. simulated shocks.

Output:

- Y : The forecast of the endogenous variables as a 1 x nSteps double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_conditionalSecondOrderMoments

```
c = nb_conditionalSecondOrderMoments(A,B,C,vcv,varX,nSteps)
c = nb_conditionalSecondOrderMoments(A,B,C,vcv,varX,nSteps,...  
type,depInd)
```

Description:

Calculate conditional second order moments of a model on a state space form.

$$y(t) = A(s)y(t-1) + B(s)x(t) + C(s)e(t)$$

where:

$$\begin{aligned}x(t) &\sim N(0, varX) \\ e(t) &\sim N(0, vcv(s))\end{aligned}$$

Input:

- A : See above.
- B : See above.
- C : See above.
- vcv : See above.
- varX : See above.
- Q : Transition probability matrix. If this is not empty the A, B, C and vcv inputs must be of class cell.

- nSteps : Numbr of steps ahead to calculate the condition second order moments.
- type : Either 'covariance' (default) or 'correlation'.
- depInd : Index of the endogneous variables you want the second order matrix of. As a 1 x N integer. Defualt is to return the covariance/correlation matrix of all endogneous variables.

Output:

- c : A nVar*nSteps x nVar*nSteps double.

See also:

[nb_forecast.buildShockRestrictions](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_defaultWeights

`weights = nb_defaultWeights(scores, varargin)`

Description:

Convert scores into weights using the formula:

`weight(h, v, c, i) = score(h, v, c, i) / sum_i(score(h, v, c, i))`

where h is the horizon, v is the variable, c is the context and i is the model.

Input:

- scores : A nHor x nVar x nContexts x nModel double.

Optional input:

- 'num' : Only keep X number of models. Must be a scalar integer > 0. If this number is larger than the number of models, all models are selected. The rest of the model is weights as described.
- 'perc' : Only keep the X% best models. Must be a scalar double between 1 and 100. Caution: Will use the ceil operator, to ensure that the number of selected models are strictly positive and an integer. The rest of the model is weights as described. Must be a scalar integer > 0.
- 'remove' : Remove X number of models. Must be a scalar integer > 0. If this number is larger than the number of models, one model is selected.

Output:

- weights : A nHor x nVar x nContexts x nModel double.

See also:

[nb_model_group_vintages.constructWeights](#)
[nb_model_group_vintages.combineForecast](#)

Written by Kenneth Å!terhagen Paulsen

◆ nb_dieboldMarianoTest

```
[test,pval] = nb_dieboldMarianoTest(y1,y2,actual,bandWidth,...  
    bandWidthCrit)  
[test,pval] = nb_dieboldMarianoTest(y1,y2,actual,bandWidth,...  
    bandWidthCrit,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Implements section of Timmermann and Zhu (2019), Comparing Forecasting Performance with Panel Data when multivariate is set to true.

Caution: Handle nan values in y1, y2 and actual.

Input:

- y1 : Forecast of model 1. A double with size nobs x nvar x nhor.
- y2 : Forecast of model 2. A double with size nobs x nvar x nhor.
- actual : Actual data. A double with size nobs x nvar.
- bandWidth : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the bandWithCrit input. Must be set to an integer greater then 0.
- bandWidthCrit : Band width selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.

Optional inputs:

- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : Test statistic. As a nhor x nvar (1) double.
- pval : P-value of test statistic. As a nhor x nvar (1) double.

See also:

Written by Kenneth Sætherhagen Paulsen

◆ nb_estimateKernelDensity

```
density = nb_estimateKernelDensity(Y,int,varargin)
```

Description:

Get kernel density estimate given forecast and evalutaion points

Input:

- Y : A nHor x nVar x nDraws double with the simulated forecasts.
- int : The output from nb_getDensityPoints
- varargin : Optional inputs given to the nb_ksdensity function

Output:

- density : A 1 x nVar cell with the stored densities of the forecast.

See also:

[nb_getDensityPoints](#), [nb_evaluateDensityForecast](#), [ksdensity](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_evaluateDensity

```
fcstEval = nb_evaluateDensityForecast(type,data,density,domain,...  
                                         meanForecastData,forecastData,dist)
```

Description:

Evaluate density forecast given a density and mean forecast.

Input:

```

- type          : Type of evaluation (Could be a cellstr with more
                  of these):
                  - 'se'           : Square error
                  - 'abs'          : Absolute error
                  - 'diff'         : Forecast error
                  - 'logscore'     : Log score

- data          : The true data to be evaluated against. As a nobs x
                  nvar double.

- density       : A 1 x nVars x nPeriods cell. Each element is a double
                  with size nHor x nDomain.

- int           : A 1 x nVars x nPeriod cell. Each element is a double
                  with size 1 x nDomain.

- meanForecastData : A double with size nobs x nvar with the mean
                  forecast. If empty it will be calculated.

- forecastData  : A double with size nobs x nvar x nsim with the
                  density forecast.

- dist          : A nHor x nVars x nPeriods nb_distribution object.
                  If this input is provided, these distributions will
                  be the basis for the calculated log scores.

```

Output:

```

- fcstEval : A Struct with the following properties:

    > (x)      : The evaluation score. As a nHor x nvar
                  double.

    > density : A 1 x nVar cell. Each element consist of nHor
                  x nPoints. For each variable the values of
                  the density at the given points is stored.

    > int      : A 1 x nVar cell. Each element consist of nHor
                  x nPoints. For each variable the points where
                  the density is evaluated is stored.

```

Written by Kenneth Sæterhagen Paulsen

◆ nb_evaluateDensityForecast

```
fcstEval = nb_evaluateDensityForecast(type,data, ...
                                       forecastData,meanForecastData,inputs)
```

Description:

Evaluate density forecast given a simulated density and mean forecast.

Input:

```

- type          : Type of evaluation (Could be a cellstr with more
                  of these):
                  - 'se'           : Square error
                  - 'abs'          : Absolute error
                  - 'diff'         : Forecast error
                  - 'logscore'     : Log score

- data          : The true data to be evaluated against. As a nobs x
                  nvar double.

- forecastData : A double with size nobs x nvar x nsim with the
                  density forecast.

- meanForecastData : A double with size nobs x nvar with the mean
                  forecast. If empty it will be calculated.

- inputs        : A struct with the following fields (You don't need
                  to provide this input):
                  > 'bins'          : See the 'bins' option of the
                                      uncondForecast method of the
                                      nb_model_generic class.
                  > 'estDensity'    : See the 'estDensity' option of the
                                      uncondForecast method of the
                                      nb_model_generic class.

```

Output:

```

- fcstEval : A Struct with the following properties:
              > <type>   : The evaluation score. As a nHor x nvar
                            double.

              > density   : A 1 x nVar cell. Each element consist of
                            nHor x nPoints. For each variable the
                            values of the density at the given points
                            is stored.

              > int       : A 1 x nVar cell. Each element consist of
                            nHor x nPoints. For each variable the
                            points where the density is evaluated is
                            stored.

```

Written by Kenneth Sæterhagen Paulsen

◆ nb_evaluateDensityForecastOnlyScore

```
score = nb_evaluateDensityForecastOnlyScore(type,data,forecastData,...  
meanForecastData)
```

Description:

Evaluate density forecast given a simulated density and mean forecast.

Input:

- type : Type of evaluation (Could be a cellstr with more of these):
 - 'se' : Square error
 - 'abs' : Absolute error
 - 'diff' : Forecast error
 - 'logscore' : Log score (Based on a normal distribution)
- data : The true data to be evaluated against. As a nobs x nvar double.
- forecastData : A double with size nobs x nvar x nsim with the density forecast.
- meanForecastData : A double with size nobs x nvar with the mean forecast. If empty it will be calculated.

Output:

- fcstEval : The evaluation score. As a nHor x nvar double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_evaluateForecast

```
score = nb_evaluateForecast(type,data,forecastData)
```

Description:

Evaluate

Input:

- type : Type of evaluation:
 - 'se' : Square errors
 - 'abs' : Absolute error
 - 'diff' : Forecast error
- data : The true data to be evaluated against. As a nobs x nvar double.
- forecastData : A double with size nobs x nvar.

Output:

- score : The evaluation score. As a nSteps x nvar double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_forecast

```
forecastDB = nb_forecast(model,options,results,startInd,endInd,nSteps,...  
                         inputs,condDB,condDBVars,shockProps)
```

Description:

Do conditional forecast with a model written on a companion form.

See Junior Maih (2010), Conditional forecast in DSGE models and Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

This function makes it also possible to use arbitrary distribution to draw from to make uncertainty bands:

1. It is possible to identify the shock given the conditional information and then using the shockProps input to specify the distributions to draw from around the identified shocks. It is important that the mean of the specified distributions are the same as the identified shocks!
2. It is possible to use the condDB input to provide the distributions to condition on. Then by using the correlation matrix provided by inputs.sigma and a gaussian copula random draws are made from the multivariate conditional distribution. For each draw the shocks to match the conditional information are found and the conditional forecasts for that draw is produced. This will map the distribution of the conditional information into restrictions on the distribution of the shocks. You should check the distribution of the shocks after using this algorithm to assure you that the identification has succeeded.

Input:

- model : See the solution property to the model object being forecasted. E.g. the property solution of the nb_model_generic class.
- options : Estimation options. Output from the estimator functions. E.g. nb_olsEstimator.estimate.
- results : Estimation results. Output from the estimator functions. E.g. nb_olsEstimator.estimate.
- startInd : The start period of the forecast. (I.e. history + 1)
- endInd : If recursive conditional forecast is wanted this period must be provided. Be aware that the condDB input must then have its third dimension set to:

nPeriods = endInd - startInd + 1

If empty nPeriods = 1.
- nSteps : The number of steps to forecast. Default is 8.

- inputs : A struct with fields (See nb_model_generic.forecast)
- condDB : One of the following:
 - > A double with size nSteps x nCondVar x nPeriods with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A double with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on.

If you condition on endogenous variables (only) and no residuals/shocks this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced or forecast condition on exogenous variables set to zero (Except seasonals, constant and time-trend)
- condDBVars : A cellstr with the variables to condition on. Can include endogenous, exogenous and shock variables.
- shockProps : A 1xnExo struct with fields:
 - > Name : The name of the residual/shock to use to match the conditional information. If an exogenous variable is not included here, it means that it is not activated.

```

> StandardDeviation : Standard deviation of the shock.
    This option can be used to adjust
    the standard deviation away from the
    estimated one. If given as nan the
    estimated one is used.

> Horizon           : Anticipated horizon of the shocks.

> Periods            : The active periods of the shock. If
    nan, the shock is active for all
    periods.

```

Output:

- forecast : A struct with the forecast output.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getDensityPoints**

```
int = nb_getDensityPoints(Y,bins,scale)
```

Description:

Get selected bins of the distribution. I.e. the stored points of the distribution

Input:

```

- Y      : A nHor x nVar x nDraws double with the simulated forecasts.

- bins   : The length between the points of the distribution.

> []      : The domain will be found. (default is that 1000
    observations of the density is stored).

> integer : The min and max is found but the length of the bins
    is given by the provided integer.

> cell    : Must be on the format:

    {indVar1,lowerLimit1,upperLimit1,binsL1;
     indVar2,lowerLimit2,upperLimit2,binsL2;
     ...}

- lowerLimit1 : An integer, can be nan, i.e. will
    be found.

- upperLimit1 : An integer, can be nan, i.e. will
    be found.

- binsL1      : An integer with the length of the
    bins. Can be nan. I.e. bins length
    will be adjusted to create a
    domain of 1000 elements.

```

```
- scale : Number of standard deviation to add at ends of the domain.  
Default is 2;
```

Output:

```
- int : A 1 x nVar cell with the selected evaluation points of the  
density. (Which is yet to be found, see  
nb_estimateKernelDensity)
```

See also:

[nb_estimateKernelDensity](#), [nb_evaluateDensityForecast](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_hdi

```
ci = nb_hdi(data)  
ci = nb_hdi(data,limits,dim,varargin)
```

Description:

Construct credibility interval by the highest density interval method

Input:

```
- data : A dim1 x dim2 x dim3 double. Can include nan values  
- limits : The wanted limits, as a 1 x nLim. E.g:  
    - 0.9      : Gives the 90 percent highest density interval.  
                  (Default)  
    - [0.1,0.2] : Gives the 10 and 20 percent highest density  
                  intervals.  
- dim : The dimension to calculate the highest density interval over.  
Default is 1.
```

Optional inputs:

```
- varargin : Optional input given to the nb_ksdensity function.
```

Output:

```
- ci : A double where the dimension given by dim has been changed to  
match the number of found intervals (these may be more than  
nLim*2, as this measure over intervals may be  
non-overlapping).  
- limInd : A double matching the return number of intervals. If  
ci is a dim1 x dim2 x rLim, then this output is 1 x 1 x rLim  
cellstr.
```

Examples:

```
[ci,limInd] = nb_hdi(randn(100,1),[0.1,0.2],1)
```

See also:

[nb_ksdensity](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_identifyShocks**

```
E = nb_identifyShocks(A,B,C,Y0,X,E,MUy,indY)
```

Description:

Compute forecast of equations on the form;

```
Y(:,t) = A*Y(:,t-1) + B*X(:,t) + C*E(:,t)
```

But where the $Y(:,t)$ for each iteration is set to its conditional restrictions.

Input:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nresidual double.
- Y0 : A nendo x 1 double, with the initial values.
- X : A nexo x nSteps double with the exogenous forecast data.
- E : A nres x nSteps double with the residual data. E.g. simulated shocks.
- MUy : A nvar (<nendo) x nSteps double.
- indY : The location of MUy in Y. First dimension.

Output:

- E : Identified shocks, as nShock x nSteps double.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_irfEngine**

```
[irfData,paused] = nb_irfEngine(solution,options,inputs)
```

Description:

Calculate irfs (with error bands) with a model written on a companion form.

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- solution : A struct storing the companion form of the model. See the help on the property nb_model_generic.solution for more on this input.
- options : Estimation options. Output from the estimator functions. E.g. nb_olsEstimator.estimate.
- results : Estimation results. Output from the estimator functions. E.g. nb_olsEstimator.estimate.
- inputs : A struct with different inputs. See nb_model_genric.irf.

Output:

- irfData : If perc is empty; A periods + 1 x nVar x nShocks otherwise; A periods + 1 x nVar x nShocks x nPerc + 1, where the mean will be at irfData(:,:, :, end)
- paused : true or false

Written by Kenneth Sætherhagen Paulsen

◆ **nb_kernelCDFBounds**

```
out = nb_kernelCDFBounds(type)
```

Description:

See output.

Input:

- type : 1: Lower. 0: Upper.

Output:

- out : Upper or lower bound on CDF estimation by kernel methods.

See also:

[nb_evaluateDensityForecast](#), [nb_simulateFromDensity](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_kolsrudJointPredictionBands**

KJPB = nb_kolsrudJointPredictionBands(fcst,percUL)

Description:

Calculate joint prediction bands due to Dag Kolsrud (2015):
"A time-simultaneous prediction box for a multivariate time series", Journal of Forecasting.

Input:

- fcst : A nHor x nVar x nDraws double
- percUL : A 1 x nPerc double with the percentiles to calculate.
E.g. [5,15,25,35,65,75,85,95]

Output:

- KJPB : A nHor x nVar x nDraws double.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_makeArtificial**

YDRAW = nb_makeArtificial(model,options,results,method,replic)

Description:

Make artificial data using bootstrap methods.

See also:

[nb_irfEngine](#), [nb_uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_mincerZarnowitz**

[test,pval] = nb_mincerZarnowitz(actual,predicted)

Description:

Do the Mincer-Zarnowitz test for bias in the forecast from a model.

```
actual = beta_0 + beta_1*predicted + e
```

Caution: Handles nan values in the predicted and the actual inputs!

Input:

- actual : A nobs x nvar double with the actual data to compare against.
- predicted : A nobs x nvar x nhor double with the predictions at horizon 1:nhor for observation 1:nobs.

Output:

- test : The F-test statistic of $\beta_0 = 0$ and $\beta_1 = 0$. Will be of size nhor x nvar.
- pval : The p-value of the test statistic. Found from the $F(2, nobs-2)$ distribution. Will be of size nhor x nvar.

See also:

[nb_uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_restrictedWeights

```
weights = nb_restrictedWeights(scores, limits, selectors)
```

Description:

Convert scores into weights using the formula:

```
weight(h,v,c,i) = score(h,v,c,i)/sum_i(score(h,v,c,i))
```

but were the number of kept models are given some criteria and dependent on the number of models that are being weighted.

Input:

- scores : A nHor x nVar x nContexts x nModel double.
- limits : A 1 x N - 1 double with the limits of the N selectors. Default is [10,100]. The limits apply to the number of models. By the default we use selector 1, if number of models are less than 10. Selector 2 if models are less than 100, and selector 3 if greater or equal to 100.
- selectors : A 1 x N cell array with the selectors for each interval. The selectors must take one input, and return the number of selected numbers out of the number provided models. Default is {@(x)max(3,round(x*0.5)), @(x)max(5,round(x*0.2)), @(x)20}

Output:

- weights : A nHor x nVar x nContexts x nModel double.

See also:

[nb_model_group_vintages.constructWeights](#)
[nb_model_group_vintages.combineForecast](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_scoreType2evalType

[evalType,type] = nb_scoreType2evalType(type)

Description:

Get evaluation type from score type.

Input:

- type : Score type, as a one line char. May include a frequency frequency postfix (_4 or _12).

Output:

- evalType : Evaluation type, as a one line char. With frequency postfix.
- type : Score type, as a one line char. Without frequency postfix.

Written by Kenneth Sæterhagen Paulsen

◆ nb_simulateFromDensity

Y = nb_simulateFromDensity(density,int,nDraws)

Description:

Simulate from known densities.

Inspired by the function density2Simulation from the PROFOR Team.

Input:

- density : One of:
> A 1 x nVars x nPer cell. Each element consist of a nHor x nDomain double storing the densities evaluated at a domain with nDomain elements at all forecasting horizons nHor. The dimension 2 of each element of the density cell matrix must match match the dimension 2 of each element of the cell matrix int.

```

> A nHor x nDomain x nPer double (int must also be a double).

- int      : A 1 x nVars x nPer cell. Each element must store a 1 x
              nDomain or a nHor x nDomain double, storing the domain of
              the density located at the same location in the density
              input.
> A nHor x nDomain x nPer double.

- nDraws   : An integer. Default is 1000.

```

Output:

```

- Y       : Dependent on input:
> cell    : A nHor x nVars x nPer x nDraws double.
> double  : A nHor x nDraws x nPer

```

See also:

[nb_model_group.combineForecast](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_varDecomp**

```
decomp = nb_varDecomp(model,options,results,inputs)
```

Description:

Variance decomposition of a model on the companion form.

Input:

```
- model    : A struct storing the companion form of the model:
```

```
    y_t = A*y_t_1 + B*x_t + C*e_t
```

Fields:

```
> A      : See the equation above. A nendo x nendo
            double.
```

```
> B      : See the equation above. A nendo x nexo
            double.
```

```
> C      : See the equation above. A nendo x nresidual
            double.
```

```
> endo : A cellstr with the decleared endogenous
            variables.
```

```
> exo  : A cellstr with the decleared exogenous
            variables.
```

```
> res  : A cellstr with the decleared
```

residuals/shocks.

```

> vcv   : Variance/covariance matrix of the
           residuals/shocks.

- options   : Estimation options. Output from the estimator functions.
              E.g. nb_olsEstimator.estimate.

- results   : Estimation results. Output from the estimator functions.
              E.g. nb_olsEstimator.estimate. Need the residual to
              bootstrap.

- inputs    : See nb_model_genric.variance_decomposition.

```

Output:

```

- decomp : If perc is empty; A nHor x nShocks + 1 x nVar
           otherwise; A nHor x nShocks + 1 x nVar x nPerc + 1,
           where the median will be at decomp(:,:, :, end).

```

Caution: The residual variance will be at decomp(:, end, :, :)

Caution: See the input output!

Written by Kenneth Sæterhagen Paulsen

25.18 Models

- nb_arima
- nb_calculate_expr
- nb_calculate_hp
- nb_compareCoeffs
- nb_dsge
- nb_estimateDensityForforecast
- nb_favar
- nb_fmdyn
- nb_forecast2Excel
- nb_isModelMarkovSwitching
- nb_midas
- nb_model_group
- nb_model_selection_group
- nb_plotPPAMultipliers
- nb_sa
- nb_var
- nb_calcMultiplier
- nb_calculate_factors
- nb_calculate_seasonal
- nb_COMPARESteadystate
- nb_ecm
- nb_exprModel
- nb_fm
- nb_fmsa
- nb_getModelNames
- nb_mfvar
- nb_model_convert
- nb_model_recursive_detrending
- nb_nonLinearEq
- nb_rw
- nb_singleEq

■ nb_arima

Go to: [Properties](#) | [Methods](#)

A class for estimation of ARIMA models.

Superclasses:

nb_model_generic

Constructor:

obj = nb_arima(varargin)

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_arima object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|---------------|----------------------|-------------|------------------|-------------|
| • addAutoName | • addAutoNameIfLocal | • addID | • addIDIfLocal | • dependent |
| • endogenous | • estOptions | • exogenous | • forecastOutput | • name |
| • options | • parameters | • reporting | • residuals | • results |
| • solution | • transformations | • userData | | |

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.

As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- `getScore`
- `getVariablesList`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`
- `removeObservations`
- `shock_decomposition`
- `simulateFromDensity`
- `solve`
- `solveNormal`
- `solveVector`
- `template`
- `testParameters`
- `uncertaintyDecomposition`
- `unstruct`
- `variance_decomposition`
- `getSolution`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `irf`
- `isFiltered`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `set`
- `simulate`
- `simulatedMoments`
- `solveARIMAEq`
- `solveRecursive`
- `struct`
- `testForecastRestrictions`
- `theoreticalMoments`
- `uncondForecast`
- `update`

► **`appendData`** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_arima.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_arima.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_arima.forecast](#), [nb_arima.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_arima.forecast`, `nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast`, `nb_arima.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_arima.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'          : The length of the bins og the domain of the density.
                    Either;
                      > []      : The domain will be found. See
                        nb_getDensityPoints (default is that 1000
                        observations of the density is stored).
                      > integer : The min and max is found but the length
                        of the bins is given by the provided
                        integer.
                      > cell     : Must be on the format:
                        {'Var1',lowerLimit1,upperLimit1,binsL1;
                         'Var2',lowerLimit2,upperLimit2,binsL2;
                         ...}
                      - lowerLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - upperLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - binsL1      : An integer with the
                        length of the bins. Can
                        be nan. I.e. bins length
                        will be adjusted to
                        create a domain of 1000
                        elements.

Caution : The variables not included will be given
           the default domain. No warning will be
           given if a variable is provided in the
           cell but not forecast by the model. (This
           because different models can forecast
           different variables)

Caution : The combineForecast method of the
           nb_model_group class is much faster if the
           lower limit, upper limit! Or else
           it must simulate new draws and do kernel
           density estimation for each model again
           with the shared domain of all models
           densities.

- 'startDate'    : The start date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
         the start date of the recursive forecast.
         Default is to start from the first possible
         date.

- 'endDate'      : The end date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).
- All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
- > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
- > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
 the ergodic steady-state for
 MS-model, and in the last regime
 for break-point models.

- 'startingProb' : Either a double or a char:
 > 'zero' : Start from zero on all variables
 (Except for the deterministic
 exogenous variables)

> [] : If the model is filtered the
 starting value is calculated
 on filtered transition
 probabilities. Otherwise the
 ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x
 nRegime double. nPeriods refer to
 the number of recursive periods to
 forecast.

> 'ergodic' : Start from the ergodic transition
 probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws
 that give rise to an unstable model. Default
 is false.

- 'states' : Either an integer with the regime to
 forecast/simulate in, or an object of class nb_ts
 with the regimes to condition on. The name of the
 variable must be 'states'.

Caution: For break-point models this is decided by
 the dates of the breaks, and cannot be set
 by this option!

- 'compareToRev' : Which revision to compare to. Default is final
 revision, i.e. []. Give 5 to get fifth revision.
 must be an integer. Keep in mind that this number
 should be larger than the nSteps input.

- 'compareTo' : Sometime you want to evaluate the forecast of one
 variable against another variable which is not
 part of the model. E.g. if you use the first release
 of a variable and want to compare it against the
 final release. To do this you can give a cellstr
 with size n x 2, where n is the number of variables
 to change the the actual observations to test
 against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
 kernel density estimation of the density
 forecast.

- 'exoProj' : Tolerate missing exogenous variables by projecting
 them by a fitted model. Only when the 'condDB'
 input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_arima.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_arima.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_arima.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_arima.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsgen.filter](#), [nb_arima.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_arima.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getForecastVariables ↑](#)

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as

fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

`nb_model_generic.forecast`, `nb_model_generic.constructScore`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getResidual** ↑

`residual = getResidual(obj)`

Description:

Get residual from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- `residual` : Either a `nb_ts` or a `nb_cs` object with `residual(s)` stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualGraph** ↑

`plotter = getResidualGraph(obj)`

Description:

Get residual graph from the given estimator. The returned will be an object of class `nb_graph`, which you can call the `graphInfoStruct` method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- `obj` : An object of class `nb_model_generic` (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph`. Use the `graphInfoStruct` method or the `nb_graphInfoStructGUI` class.

Examples:

```
plotter = getResidualGraph(obj);
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of `nb_model_generic` object

Input:

- `obj` : A vector of `nb_model_generic` objects

Output:

- `residualNames` : A cellstr with the unique residual names of all the models

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method `solve`.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
- 'RMSE' : One over root mean squared error
- 'MSE' : One over mean squared error
- 'MAE' : One over mean absolute error
- 'MEAN' : One over mean error
- 'STD' : One over standard error of the forecast error
- 'ESLS' : Exponential of the sum of the log scores
- 'EELS' : Exponential of the mean of the log scores
- 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.

- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_arima.simulatedMoments](#), [nb_arima.theoreticalMoments](#)
[nb_arima.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_arima.help
helpText = nb_arima.help(option)
helpText = nb_arima.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **initialize ↑**

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given,

otherwise this option does not apply.

- 'optimizer'
 - : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'
 - : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'
 - : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'
 - : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods'
 - : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'
 - : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'
 - : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'
 - : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'
 - : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights'
 - : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_arima.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgc objects.
- 'compare' : Give true if you have a scalar nb_dsgc model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

```

- 'draws' : Number of draws for calculating girf. Default is
           1000. For Markov switching models this will set the
           number of simulated paths of states. For more see
           the 'type' input.

- 'factor' : A cell array with the factors to multiply the
             irf of the individual model variables.
             I.e. {'var1',100,...} or
                  {'var1','var2'},100,...}

If the string starts with a asterisk you will
multiply all the variables that contain that
string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This option sets the error band percentiles of the
               graph, when the 'perc' input is empty. As a 1 x
               numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
               Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from
                  this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

    > 'cumulative product' : cumprod(1 + X,1)

    > 'cumulative sum' : cumsum(X,1)

    > 'cumulative product (log)' : log(cumprod(1 + X,1))

    > 'cumulative sum (exponential)' : exp(cumsum(X,1))

    > 'cumulative product (%)' : cumprod(1 + X/100,1)

    > 'cumulative sum (%)' : cumsum(X/100,1)

    > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))

    > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))

    > '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
             Default is ''. See help on the method input of the
             nb_model_generic.parameterDraws method for more
             this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
                    the map from structural shocks to dependent

```

variables. See `nb_var.set_identification`. Of course this option only makes sense for VARs identified with sign restrictions.

- `'normalize'` : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to `inf`, it will normalize the max impact period.
E.g. `{'Var',1,2}, {'Var',1,inf}`
Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- `'normalizeTo'` : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- `'newReplic'` : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- `'parallel'` : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- `'parallelL'` : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.
- `'pause'` : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- `'perc'` : Error band percentiles. As a 1 x `numErrorBand` double. The input must be given as the coverage, and not the percentiles itself. E.g. `[0.3,0.5,0.7,0.9]`. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So `[0.3,0.5,0.7,0.9]` will get the percentiles `[5,15,25,35,65,75,85,95]`, i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- `'periods'` : Number of periods of the impulse responses.
- `'plotSS'` : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.

```

- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be
a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws
that give rise to an unstable model. Default
is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar : Select the the regime to take the
initial transition probabilities
from.

    > double : A draws x nRegime or a 1 x
nRegime double. draws refer to
the number set by the 'draws'
input.

    > 'ergodic' : Start from the ergodic transition
probabilities. Default for 'irf'.

    > 'random' : Randomize the starting values
using the simulated starting
points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing

```

with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

```

> 'zero'          : Start from zero on all variables.

> 'random'        : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'         : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'           : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'   : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'       : The variables to create impulse responses of.
                     Default is the dependent variables defined by the
                     first model.

Caution: The variables reported by the reporting
          property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                     using the method specified by 'levelMethod'.

```

Output:

```

- irfs      : The (central) impulse response function stored in a
               structure of nb_ts object. Each field stores the the
               impulse responses of each shock. Where the variables
               of the nb_ts object is the impulse responses of the
               wanted endogenous variables.

               I.e. the impuls responses to the shock 'E_X_NW' is
               stored in irfs.('E_X_NW'). Which will then be a nb_ts
               object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
          responses from each model are saved as
          different datasets (pages) of the nb_ts object.

```

```

Caution: Each nb_model_generic object can have different
variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store
  a nb_ts object with the error bands of all variables. The
  pages of the nb_ts object is the percentiles (from lower
  to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter   : An object of class nb_graph_ts.

  > 'compareShocks' set to false (default)
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

  > 'compareShocks' set to true
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

- obj       : If the process has been paused, some temporary output will
  be stored in the object, and if you want to continue at a
  later stage you can take up from this point using this
  returned output. If not paused this output is empty.

```

See also:

[nb_arima.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint

```
prediction bands for  
macroeconomic risk management  
  
> 'copula' : Using a copula likelihood approach.  
  
> 'mostlik' : Group the paths based on how  
likely they are.  
  
- 'nSteps' : Number of forecasting steps to use when constructing the  
error bands. If empty (default) all forecasting steps stored  
in the object is used.
```

Output:

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.
- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend
on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_arima.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. `@(x)gt(x,0), @(x)gt(x,0)||lt(x,1)`, i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. `@(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. `{'Var1','Var1',1,@(x)gt(x,0.1)}`, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. `{'Var1','Var2',0,@(x)lt(x,0.1)}`, to

test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.
 E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:
 1. The expression using any of the endogenous variables of the model.
 2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_arima.monteCarloFiltering](#), [nb_arima.irf](#)
[nb_arima.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_arima.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.

- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_arima.mcfRestriction](#), [nb_arima.solve](#)
[nb_arima.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'copulaBootstrap' : Uses a copula approach to draw residual that are autocorrelated, Does not handle heteroscedasticity. Only an option for models estimated with classical methods.
 - > 'posterior' : Posterior draws. Only for models estimated with bayesian methods.

Default for models estimated with
bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more on this option.
- stable : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
- lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
- factors : Draws of the factors as a T x nFac x

```

draws double.

> 'solution' : Get the solution of the model for each draw from the
distribution of parameters. The output will be a struct
with size 1 x draws. The struct will have the same
format as the solution property of the underlying
object. I.e. only one output!

> 'object'   : Get the draws as a 1 x draws nb_model_generic object.
Each element will be a model representing a given draw
from the distribution of the parameters. I.e. only one
output!

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + ( +/- nb_distribution.t_icdf(alpha/2) ) * stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```

plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)

```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► plotMCF ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot.

'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- `plotter` : > 'allInOne' : A `nb_graph_cs` object. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
 > 'biplot' : A vector of `nb_graph_data` objects with size equal to the number of pairwise combination of the parameters that can be made. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_arima.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFDistTest ↑**

`plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)`

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `test` : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_arima.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `nameValue` : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_arima.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_arima.getPosteriorDistributions](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_arima.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties
of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

[decomp, decompBand, plotter] = shock_decomposition(obj, varargin)

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and
the names of the groups. If you have not
listed a shock it will be grouped in a
group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';  
'shock_name_1' , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with
a shock name or a cellstr array with the
shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

Caution: Two special identifiers can be used;
'Initial Conditions' and 'Steady-state'. The
first represent the impact of all shocks that
hit the system before the decomposition/
estimation started. The second summarizes the
impact of steady-state changes on the system,

i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. [0.3, 0.5, 0.7, 0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.

- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_arima.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be

a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the

simulation of.

- > 'endo' : All the endogenous variables are returned.
- > 'fullendo' : All the endogenous variables are returned included the lag variables.
- > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
- > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

- > double : A 1 x nVar double.
- > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
- > 'steadystate' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsge models.

```

> 'zero'           : Start from zero on all variables
                      (Except for the deterministic
                      exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'startingProb'  : Either a double or a char:

    > double        : A nPeriods x nRegime or a 1 x
                      nRegime double. nPeriods refer to
                      the number of recursive periods to
                      forecast.

    > 'ergodic'     : Start from the ergodic transition
                      probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity ↑**

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- `obj` : An object of class nb_model_generic.
- Optional inputs;
- `'output'` : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when 'stacked' is set to true.
- `'vars'` : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- `'type'` : Either 'covariance' or 'correlation'. Default is 'correlation'.
- `'draws'` : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- `'pDraws'` : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- `'perc'` : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- `'nSteps'` : Number of simulation steps. As a 1x1 double. Default is 100.
- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting

values of the simulation. Default is 0.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Default is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_arima.graphCorrelation](#), [nb_arima.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

`obj = solve(obj)`

Description:

Solve estimated model(s) represented by nb_arima object(s).

Input:

- obj : A vector of nb_arima objects.

Output:

- obj : A vector of nb_arima objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveARIMAEq** ↑

tempSol = nb_arima.solveARIMAEq(results, opt)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

tempSol = nb_arima.solveNormal(results, opt)

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

tempSol = nb_arima.solveRecursive(results, opt)

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

obj = solveVector(obj)

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_arima.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

options = nb_arima.template()

Description:

Construct a struct which can be provided to the nb_arima class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► testForecastRestrictions ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.

> variable : The variable(s) to test,
as a string or cell array of strings.

> date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.

> test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillty intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is

```

'correlation'.

- 'pDraws'      : Number of parameter draws. Default is 1. I.e. to use
                   the estimate parameters.

- 'perc'        : Error band percentiles. As a 1 x numErrorBand double.
                   E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
                   all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method'       : The selected method to create confidenc/probability
                   bands. Default is ''. See help on the method input of
                   the nb_model_generic.parameterDraws method for more
                   this input.

- 'foundReplic'  : A struct with size 1 x pDraws. Each element
                   must be on the same format as obj.solution. I.e.
                   each element must be the solution of the model
                   given a set of drawn parameters. See the
                   parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
         to the number of draws you want to do.

```

Output:

```

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar
                 x nVar nb_cs object or double. The variance is along
                 the diagonal. (Will be symmetric)

- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar
                 x nVar nb_cs object or double. Along the diagonal is the
                 auto-covariance/correlation with the variable itself. In
                 the upper triangular part the you can find cov(x,y(-i)),
                 where x is the variable along the first dimension. In
                 the lower triangular part the you can find cov(x(-i),y),
                 where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find
      cov(x,y(-i)) you can use
      getVariable(varargout{i}, 'y', 'x'),
      while to get cov(x(-i),y) (== cov(x,y(+i))) you
      can use getVariable(varargout{i}, 'x', 'y'). (This
      example only works if the output is of class nb_cs)

```

See also:

[nb_arima.graphCorrelation](#), [nb_arima.parameterDraws](#)

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

obj = uncondForecast(obj,nSteps,varargin)

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_arima.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [unstruct](#) ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_arima.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update](#) ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type : > '' : Only update data. Default
 > 'estimate' : Update data and estimate.
 > 'solve' : Update data, estimate and solve
 > 'forecast' : Update data, estimate, solve and forecast.

- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp, decompBands, plotter, plotterBands] = ...
    variance_decomposition(obj, varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8, inf].

- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and

the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_arima.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_calculate_expr

Go to: [Properties](#) | [Methods](#)

A class for evaluation expression involving a set of series.

Constructor:

```
obj = nb_calculate_expr(varargin)
```

Optional input:

- See the set method for more on the inputs.

(nb_model_estimate.set)

Output:

- obj : A nb_calculate_expr object.

See also:

[nb_calculate_vintages](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [dependent](#)
- [estOptions](#)
- [name](#)
- [options](#)
- [reporting](#)
- [results](#)
- [transformations](#)
- [userData](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) that you want to seasonally adjust, the tex_name holds the names in the tex format. The number field holds a double with the number of variables. To set it use the set function. E.g.
obj = set(obj,'dependent',{'Var1','Var2'});
or use the <className>.template() method.

- **[estOptions](#)** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined

functions, as long as they take and return a nb_math_ts object.
A nb_math_ts object is just a double with a time dimension!
To get a list of all the methods of the nb_math_ts class use;
methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_CALCULATE_ONLY

Methods:

- appendData
- calculate
- checkReporting
- convert
- convertEach
- createVariables
- estimate
- getCalculated
- getEstimationOptions
- getModelNames
- getModelVars
- getOriginalVariables
- handleCondInfo
- handleMissing
- help
- isStatic
- print
- print_estimation_results
- removeObservations
- set
- struct
- template
- unstruct

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over
the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **calculate** ↑

calc = calculate(obj,varargin)

Description:

Do the calculation associated with the object.

Input:

- obj : An object array of class nb_calculate_generic.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- calc : An cell array of class nb_ts, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_calculate_expr.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **convert** ↑

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,    description
    'VAR1_G',  'pcn(VAR1)', 'avg',   'VAR1 growth'
    'VAR2_G',  'pcn(VAR2)', 'avg',   'VAR2 growth'
    'VAR3_G',  'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```
obj      = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **getCalculated** ↑

```
calc = getCalculated(obj)
```

Description:

Get calculated series from model.

Input:

- obj : A scalar nb_calculate_generic object.

Output:

- calc : An object of class nb_ts.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent and exogenous variables.

For factor models the observables are added as well.

Input:

- obj : A scalar nb_model_generic object.
- varsIn : Not in use!

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_ONLY

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
helpText = nb_calculate_expr.help
helpText = nb_calculate_expr.help(option)
helpText = nb_calculate_expr.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► isStatic ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► print ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_calculate_expr.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► print_estimation_results ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► removeObservations ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► set ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_calculate_expr.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

options = nb_calculate_expr.template()
options = nb_calculate_expr.template(num)

Description:

Construct a struct which can be provided to the nb_calculate_expr class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_model_estimate.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_calculate_expr.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

■ **nb_calculate_factors**

Go to: [Properties](#) | [Methods](#)

A class for calculating factors from a set of selected variables.

Constructor:

obj = nb_calculate_factors(varargin)

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_calculate_factors object.

See also:

[nb_calculate_vintages](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [estOptions](#)
- [name](#)
- [observables](#)
- [options](#)
- [reporting](#)
- [results](#)
- [transformations](#)
- [userData](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[estOptions](#)** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **[name](#)** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **[observables](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observables',{'Var1','Var2'}); or use the <className>.template() method.

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_CALCULATE_ONLY

Methods:

- appendData
- calculate
- checkReporting
- convert
- convertEach
- createVariables
- estimate
- getCalculated
- getEstimationOptions
- getModelNames
- getModelVars
- getOriginalVariables
- handleCondInfo
- handleMissing
- help
- isStatic
- print
- print_estimation_results
- removeObservations
- set
- struct
- template
- unstruct

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over
the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► calculate ↑

```
calc = calculate(obj,varargin)
```

Description:

Do the calculation associated with the object.

Input:

- obj : An object array of class nb_calculate_generic.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- calc : An cell array of class nb_ts, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_calculate_factors.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► [checkReporting ↑](#)

`obj = checkReporting(obj)`

Description:

Check reporting, and store historical observation of reported variables.

Input:

- `obj` : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
- > third column : Any expression that can be interpreted by the nb_ts.createShift method.
- > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
  Name,    input,    shift,    description
  'VAR1_G', 'pcn(VAR1)', 'avg',   'VAR1 growth'
  'VAR2_G', 'pcn(VAR2)', 'avg',   'VAR2 growth'
  'VAR3_G', 'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```
obj      = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► getCalculated ↑

calc = getCalculated(obj)

Description:

Get calculated series from model.

Input:

- obj : A scalar nb_calculate_generic object.

Output:

- calc : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **getEstimationOptions** ↑

outOpt = getEstimationOptions(obj)

Written by Kenneth Sæterhagen Paulsen

► **getModelNames** ↑

names = getModelNames(obj)

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)

Description:

Get all variables to the model. That include dependent and exogenous variables.

For factor models the observables are added as well.

Input:

- obj : A scalar nb_model_generic object.
- varsIn : Not in use!

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_ONLY

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
helpText = nb_calculate_factors.help
helpText = nb_calculate_factors.help(option)
helpText = nb_calculate_factors.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.

- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_calculate_factors.print_estimation_results](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object

- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_calculate_factors.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template ↑**

```
options = nb_calculate_factors.template()
options = nb_calculate_factors.template(num)
```

Description:

Construct a struct which can be provided to the nb_calculate_factors class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **unstruct ↑**

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_calculate_factors.struct](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

■ **nb_calculate_hp**

Go to: [Properties](#) | [Methods](#)

A class for doing HP-filtering of series.

Constructor:

obj = nb_calculate_hp(varargin)

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : A nb_calculate_hp object.

See also:

[nb_calculate_vintages](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- | | | | | |
|-------------------|----------------------|-----------|----------------|-------------|
| • addAutoName | • addAutoNameIfLocal | • addID | • addIDIfLocal | • dependent |
| • estOptions | • name | • options | • reporting | • results |
| • transformations | • userData | | | |

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) that you want to seasonally adjust, the tex_name holds the names in the tex format. The number field holds a double with the number of variables. To set it use the set function. E.g.
obj = set(obj,'dependent',{'Var1','Var2'});
or use the <className>.template() method.

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODEL DATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODEL DATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_CALCULATE_ONLY

Methods:

- appendData
 - checkReporting
 - convertEach
 - estimate
 - getEstimationOptions
 - getModelVars
 - handleCondInfo
 - help
 - print
 - removeObservations
 - struct
 - unstruct
 - calculate
 - convert
 - createVariables
 - getCalculated
 - getModelNames
 - getOriginalVariables
 - handleMissing
 - isStatic
 - print_estimation_results
 - set
 - template
-

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODELDATA](#)

► **calculate** ↑

`calc = calculate(obj,varargin)`

Description:

Do the calculation associated with the object.

Input:

- `obj` : An object array of class `nb_calculate_generic`.
- `'parallel'` : Use this string as one of the optional inputs to run the estimation in parallel.
- `'cores'` : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if `'parallel'` is given.
- `'remove'` : Give `'remove'` to remove the models that return errors from the `obj` output. This input is not stored to the `forecastOutput` property of the objects.
- `'waitbar'` : Use this string to give a waitbar during estimation. I.e. when looping over models. If `'parallel'` is used this option is not supported!
- `'write'` : Use this option to write errors to a file instead of throwing the error.

Optional input:

- `varargin` : See the the set method.

Output:

- `calc` : An cell array of class `nb_ts`, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_calculate_hp.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_CALCULATE_GENERIC](#)

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **convert** ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables ↑**

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

```

> second column : Any expression that can be interpreted
      by the nb_ts.createVariable method.

> third column : Any expression that can be interpreted
      by the nb_ts.createShift method.

> fourth column : Comments

- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is
      important that this option is higher than the number of
      forecasting/irf steps, or else the output will be nan!

```

Output:

- obj : The NxM object itself added the model variables. The
 expressions input is stored in the property
 transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending.
 Use the graphInfoStruct method or the nb_graphInfoStruct
 class to produce the graph.

Examples:

```

expressions = {%
  Name,    input,    shift,    description
  'VAR1_G', 'pcn(VAR1)', 'avg',   'VAR1 growth'
  'VAR2_G', 'pcn(VAR2)', 'avg',   'VAR2 growth'
  'VAR3_G', 'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)

```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```

obj        = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)

```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.

- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Åstergård Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **getCalculated ↑**

calc = getCalculated(obj)

Description:

Get calculated series from model.

Input:

- obj : A scalar nb_calculate_generic object.

Output:

- calc : An object of class nb_ts.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► getEstimationOptions ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth SÃ¶terhagen Paulsen

► getModelNames ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► getModelVars ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent and exogenous variables.

For factor models the observables are added as well.

Input:

- obj : A scalar nb_model_generic object.
- varsIn : Not in use!

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_CALCULATE_ONLY

► getOriginalVariables ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODELDATA

► handleCondInfo ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

```
- ret : true or false. true if the estimation settings is so that the  
model handle conditional info.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

```
- obj : A scalar nb_model_estimate object.
```

Output:

```
- ret : true or false. true if the estimation settings is so that the  
model handle missing data.
```

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_calculate_hp.help  
helpText = nb_calculate_hp.help(option)  
helpText = nb_calculate_hp.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

```
- option : Either 'all' or the name of the option you want to get the  
help on.  
  
- maxChars : The max number of chars in the printout of (approx). This  
applies to the second column of the printout. Default is  
40.
```

Output:

```
- helpText : A char with the help.
```

Written by Kenneth SÃ¶terhagen Paulsen

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_calculate_hp.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **struct ↑**

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_calculate_hp.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template ↑**

```
options = nb_calculate_hp.template()
options = nb_calculate_hp.template(num)
```

Description:

Construct a struct which can be provided to the nb_calculate_hp class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **unstruct ↑**

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_calculate_hp.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

■ **nb_calculate_only**

Go to: [Properties](#) | [Methods](#)

An abstract superclass for all classes that has a calculate method but does not inherit the nb_model_generic class.

Superclasses:

[nb_calculate_generic](#)

Constructor:

This class is abstract, and has no constructor,

See also:

[nb_calculate_generic](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- addAutoName
- addAutoNameIfLocal
- addID
- addIDIfLocal
- estOptions
- name
- options
- reporting
- results
- transformations
- userData

- **addAutoName ↑**

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Methods:

- [appendData](#)
- [convert](#)
- [estimate](#)
- [getModelVars](#)
- [handleMissing](#)
- [print_estimation_results](#)
- [struct](#)
- [calculate](#)
- [convertEach](#)
- [getCalculated](#)
- [getOriginalVariables](#)
- [isStatic](#)
- [removeObservations](#)
- [unstruct](#)
- [checkReporting](#)
- [createVariables](#)
- [getModelNames](#)
- [handleCondInfo](#)
- [print](#)
- [set](#)

► **appendData** ↑

obj = appendData(obj,DB)

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► calculate ↑

calc = calculate(obj, varargin)

Description:

Do the calculation associated with the object.

Input:

- obj : An object array of class nb_calculate_generic.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e.

when looping over models. If 'parallel' is used this option is not supported!

- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- calc : An cell array of class nb_ts, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_calculate_only.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **convert** ↑

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,     shift,      description
'VAR1_G',   'pcn(VAR1)', 'avg',    'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',    'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```
obj      = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_ESTIMATE](#)

► **getCalculated** ↑

```
calc = getCalculated(obj)
```

Description:

Get calculated series from model.

Input:

- obj : A scalar [nb_calculate_generic](#) object.

Output:

- calc : An object of class [nb_ts](#).

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_CALCULATE_GENERIC](#)

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M [nb_model_forecast](#) object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_NAME](#)

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent and exogenous variables.

For factor models the observables are added as well.

Input:

- obj : A scalar nb_model_generic object.
- varsIn : Not in use!

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_calculate_only.print_estimation_results](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print_estimation_results** ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **removeObservations ↑**

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► struct ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_calculate_only.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_calculate_only.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

■ **nb_calculate_seasonal**

Go to: [Properties](#) | [Methods](#)

A class for doing seasonal adjustment of series.

Constructor:

```
obj = nb_calculate_seasonal(varargin)
```

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_calculate_seasonal object.

See also:

[nb_calculate_vintages](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [dependent](#)
- [estOptions](#)
- [exogenous](#)
- [name](#)
- [options](#)
- [reporting](#)
- [results](#)
- [transformations](#)
- [userData](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) that you want to seasonally adjust, the tex_name holds the names in the tex format. The number field holds a double with the number of variables. To set it use the set function. E.g.
obj = set(obj,'dependent',{'Var1','Var2'});
or use the <className>.template() method.

- **[estOptions](#)** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **[exogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** [↑](#)

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_CALCULATE_ONLY

Methods:

- [appendData](#)
- [checkReporting](#)
- [convertEach](#)
- [estimate](#)
- [getEstimationOptions](#)
- [getModelVars](#)
- [handleCondInfo](#)
- [help](#)
- [print](#)
- [removeObservations](#)
- [struct](#)
- [unstruct](#)
- [calculate](#)
- [convert](#)
- [createVariables](#)
- [getCalculated](#)
- [getModelNames](#)
- [getOriginalVariables](#)
- [handleMissing](#)
- [isStatic](#)
- [print_estimation_results](#)
- [set](#)
- [template](#)

► **appendData** [↑](#)

obj = appendData(obj,DB)

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **calculate** ↑

calc = calculate(obj,varargin)

Description:

Do the calculation associated with the object.

Input:

- obj : An object array of class nb_calculate_generic.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- calc : An cell array of class nb_ts, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_calculate_seasonal.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► [checkReporting ↑](#)

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► [convert ↑](#)

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.

- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of `nb_ts.convert`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► **convertEach** ↑

`obj = convert(obj, freq, methods, varargin)`

Description:

Convert the frequency of the data of the `nb_modelData` object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► **createVariables** ↑

`obj = createVariables(obj, expressions, fcstHorizon)`
`[obj, plotter] = createVariables(obj, expressions, fcstHorizon)`

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.
 - > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,    description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```
obj          = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **getCalculated** ↑

```
calc = getCalculated(obj)
```

Description:

Get calculated series from model.

Input:

- obj : A scalar nb_calculate_generic object.

Output:

- calc : An object of class nb_ts.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent and exogenous variables.

For factor models the observables are added as well.

Input:

- obj : A scalar nb_model_generic object.
- varsIn : Not in use!

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_ONLY

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
helpText = nb_calculate_seasonal.help
helpText = nb_calculate_seasonal.help(option)
helpText = nb_calculate_seasonal.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► isStatic ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► print ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_calculate_seasonal.print_estimation_results](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print_estimation_results** ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **removeObservations** ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_ESTIMATE](#)

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class [nb_model_estimate](#).

Output:

- s : A struct representing the [nb_model_generic](#) object.

See also:

[nb_calculate_seasonal.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_ESTIMATE](#)

► **template** ↑

```
options = nb_calculate_seasonal.template()
options = nb_calculate_seasonal.template(num)
```

Description:

Construct a struct which can be provided to the [nb_calculate_seasonal](#) class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_model_estimate.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_calculate_seasonal.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

■ **nb_dsg**

Go to: [Properties](#) | [Methods](#)

If you use the 'nb_file' you will use the NB Toolbox parser and solver. How to write the model file in this case is documented in the DAG.pdf file, and there you will also find the documentation of the solution algorithm used in this case. See also the nb_dsg.solveNB method.

This class can also be used for converting RISE dsg objects and dynare structures to a nb_model_generic subclass. This makes it for example easy to compare for example IRF's from RISE and dynare with models estimated by the NB TOOLBOX, e.g. VARs.

It is also possible to give the model files directly to the nb_dsg class by using the 'rise_file' or 'dynare_file' options.

In the case the model is not parsed and solved with NB toolbox credits for solving the DSGE model should go to Junior Maih for his RISE toolbox or to the dynare development team, and not to the author of this code.

Superclasses:

`nb_model_generic, nb_model_sampling, nb_model_parse`

Constructor:

`obj = nb_dsge(varargin)`

Optional input:

- See the `nb_model_estimate.set` method for more.

See the method `assignParameters` to assign calibration, and the method `assignPosteriorDraws` to assign already found posterior draws. If the underlying object is a RISE dsge model, new draws can be made by `parameterDraws`, if the methods `irfs`, `forecast` etc. needs more draws this will be done automatically. On the other hand if `dynare` is used this is not possible.

OR

- If the first input is '`rise_file`', then the follwing can be given:

`> 'rise_file'` : The filename of the RISE model file.
Must return a dsge model. As a string.

Optional:

`> 'steady_state_file'` : The filename of the steady state file if needed. As a string.
`> 'steady_state_imposed'` : true (default) or false.

With this option the RISE model file will be parsed, and a dsge model will be returned. See the method `assignParameters` to assign calibration, or the method `estimate` to estimate the model.

OR

- If the first input is '`dynare_file`':

`> 'dynare_file'` : The filename of the dynare model file.
Must return a dsge model. As a string.

By this options dynare will be runned. This means that all options must be given to the dynare file, such as calibrated parameters or estimation of parameters. If the model is estimated the mode will be used for the parameter values, and the posterior draws will be saved for later used if they are produced.

To provide additional inputs when calling the `dynare` function, use the third input as a one line char with the additional code.

OR

- If the first input is '`nb_file`':

`> 'nb_file'` : The filename of the model file with extension `.nb` or `.mod`.

Optional:

- 'silent' : See nb_dsge.help('silent')

Use the set method to set other options found in the options structure. See nb_dsge.help for more on each option.

Caution: The observables can be set in a call to the set method,
e.g: set(obj,'observables',{'Var1','Var2'});

Output:

- obj : An nb_dsge object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_dsge.template](#), [nb_dsge.help](#)
[nb_dsge.solveNB](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [dependent](#)
- [forecastOutput](#)
- [parameters](#)
- [residuals](#)
- [transformations](#)
- [addAutoNameIfLocal](#)
- [endogenous](#)
- [name](#)
- [parser](#)
- [results](#)
- [unitRootVariables](#)
- [addID](#)
- [estOptions](#)
- [observables](#)
- [posteriorPath](#)
- [solution](#)
- [userData](#)
- [addIDIfLocal](#)
- [exogenous](#)
- [options](#)
- [reporting](#)
- [systemPriorPath](#)

- **[addAutoName](#) ↑**

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#) ↑**

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#) ↑**

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#) ↑**

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **observables** ↑

The declared observables. As a struct with fields name, tex_name and number.

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **parser** ↑

This property will store the info on the model, such as equations etc, otherwise it will be empty.

Caution: If you are dealing with an object of class nb_dsge, this property will be empty, if the DSGE model is parsed with RISE or Dynare.

Inherited from superclass NB_MODEL_PARSE

- **posteriorPath** ↑

The path to located the posterior sampling results. Use nb_loadDraws to return a struct with the posterior sampling options and output.

Inherited from superclass NB_MODEL_SAMPLING

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated

factors and/or exogenous variables of the measurement equation.

- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the declared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **systemPriorPath** ↑

The path to located the updated priors sampling results.
Use nb_loadDraws to return the output struct of the given sampler.

Inherited from superclass NB_MODEL_SAMPLING

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension!
To get a list of all the methods of the nb_math_ts class use;

```
methods('nb_math_ts').
```

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **unitRootVariables** ↑

The declared unit root variables. As a struct with fields name, tex_name and number.

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- addBreakPoint
- addEquation
- appendData
- assignPosteriorDraws
- backwardSolver
- calculateLoss
- calculateStandardError
- checkPosteriors
- checkSteadyState
- cleanSolution
- constraints
- constructScore
- convertEach
- curvature
- derivative
- doForecastPerc2Dist
- doSimulateFromDensity
- empiricalMoments
- estimate
- evalLikelihood
- evaluatePrior
- extendedPath
- findAniticipatedMatrices
- forecastPerc2Dist
- gelmanRubin
- getBalancedGrowthPath
- getDSGEVARPriorMoments
- addConstraints
- addObsModelSolver
- assignParameters
- assignTexNames
- blockDecompose
- calculateMultipliers
- checkModel
- checkReporting
- checkUpdatedPriors
- conditionalTheoreticalMoments
- constructCondDB
- convert
- createVariables
- declareUncertainParameters
- dieboldMarianoTest
- doForecastPerc2ParamDist
- doSymbolicDerivatives
- eq
- evalFestAtDates
- evaluateForecast
- expectedBreakPointSolver
- filter
- forecast
- forecastPerc2ParamDist
- getActual
- getCondDB
- getDependent

- `getDependentNames`
- `getDiagCovarianceMatrix`
- `getEstimationOptions`
- `getEstimationTable`
- `getForecast`
- `getHistory`
- `getLHSVars`
- `getModelNames`
- `getObjective`
- `getOptimalMonetaryPolicyMatrices`
- `getPIT`
- `getParameters`
- `getPredicted`
- `getRecursiveEstimationGraph`
- `getResidual`
- `getResidualNames`
- `getScore`
- `getSteadyState`
- `getVariablesList`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `interpretStochasticTrendInit`
- `isBreakPoint`
- `isFiltered`
- `isMixedFrequency`
- `isRise`
- `getDerivOrder`
- `getEndVal`
- `getEstimationStartDate`
- `getFiltered`
- `getForecastVariables`
- `getIRFMatchingFunc`
- `getLoss`
- `getModelVars`
- `getObjectiveForEstimation`
- `getOriginalVariables`
- `getParameterDrawsMethods`
- `getPosteriorDistributions`
- `getPriorDistributions`
- `getRecursiveScore`
- `getResidualGraph`
- `getRoots`
- `getSolution`
- `getUpdatedPriorDistributions`
- `geweke`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `irf`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isSampled`

- isStateSpaceModel
- isStationary
- isestimated
- issolved
- kleinSolver
- loadPosterior
- makeObservationEq
- meanPlot
- monteCarloFiltering
- objective
- optimalSimpleRules
- parameterIntervals
- parseTempParameters
- permanentShock
- plotForecastDensity
- plotMCFDistTest
- plotPosteriors
- plotUpdatedPriors
- print Cov
- print OSR
- priorPredictiveAnalysis
- removeBreakPoints
- reparse
- sample
- set
- setParametersInUse
- setRisePosteriorSettings
- isStatic
- isbayesian
- isforecasted
- jointPredictionBands
- likelihood
- looseOptimalMonetaryPolicySolver
- mcfRestriction
- mincerZarnowitzTest
- nb2RisePreparser
- optimalMonetaryPolicySolver
- parameterDraws
- parse
- perfectForesight
- plotForecast
- plotMCF
- plotMCFValues
- plotPriors
- print
- printDecisionRules
- print _estimation_ results
- rationalExpectationSolver
- removeObservations
- reportFiltered
- sampleSystemPrior
- setLossFunction
- setPrior
- setSystemPrior

- shock_decomposition
 - simulate
 - simulateFromDensity
 - simulateStructuralMatrices
 - simulatedMoments
 - solve
 - solveBalancedGrowthPath
 - solveBreakPoints
 - solveExpanded
 - solveForEpilogue
 - solveNormal
 - solveOneIteration
 - solveOneRegime
 - solveRecursive
 - solveSteadyState
 - solveVector
 - split
 - stateSpace
 - stateSpaceBreakPoint
 - stateSpaceTVP
 - stationarize
 - struct
 - structuralMatrices2JacobianNB
 - systemPriorObjective
 - template
 - testForecastRestrictions
 - testParameters
 - theoreticalMoments
 - tracePlot
 - uncertaintyDecomposition
 - uncondForecast
 - unstruct
 - update
 - updateParams
 - updateSolution
 - variance_decomposition
 - writeModel2File
 - writePDF
 - writeTex
-

► **addBreakPoint ↑**

```
obj = addBreakPoint(obj,parameters,values,date)
obj = addBreakPoint(obj,parameters,values,date,varargin)
```

Description:

To add a break in the structural parameters of the model.

Caution: To assign the parameter values the at or after the break you can also use use the nb_model_generic.assignParameters. Append the original parameter with the a subscript followed by a the date of the breakpoint, e.g. 'paramName_2012Q1'

If the 'expectedDate' option is used the model is solved with the algorithm of Kulish and Pagan (2017), Estimation ans Solution of Models with Expectations and Structural Changes.

Input:

- obj : An object of class nb_dsg.
- parameters : A cellstr with the parameters that you want to add an unexpected breakpoint to.
- values : A double with the same size as parameters with the value of the parameters at and after the break. Can be empty, if you want to use the nb_model_generic.assignParameters method to assign the values instead.
- date : Either a date string or an object of subclass of the nb_date class. This will be the date of the breakpoint.

Optional input:

- 'steady_state_exo' : A struct with the permanent shock associated with the given break point. Use nb_dsg.help('steady_state_exo') to get more help on this input. When this is added the new steady-state will be solved using numerical methods.
- 'expectedDate' : Either a date string or an object of subclass of the nb_date class. This will be the date from where the break is expected.

Optional inputs given to the nb_dsg.set method.

Output:

- obj : An object of class nb_dsg.

See also:

[nb_dsg.parse](#), [nb_dsg.set](#)

Written by Kenneth Sæterhagen Paulsen

► **addConstraints** ↑

obj = addConstraints(obj,constraints)

Description:

Add (more) constraints to the parameters during estimation. The constraints can only include parameters (and numbers).

Input:

- obj : An object of class nb_dsg or nb_nonLinearEq.
- constraints : A N x M char array or a N x 1 (or 1 x N) cellstr array. Each element will be counted as a new constraint.

Output:

- obj : An object of class nb_dsgc or nb_nonLinearEq.

See also:

[nb_dsgc.parse](#), [nb_nonLinearEq.parse](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **addEquation** ↑

obj = addEquation(obj,varargin)

Description:

Add more equations to the model.

Input:

- obj : An object of class nb_dsgc.

Optional input:

- 'endogenous' : A char or cellstr with the added endogenous variable(s).
- 'exogenous' : A char or cellstr with the added exogenous variable(s).
- 'parameters' : A char or cellstr with the added parameter(s).
- 'equations' : A char or cellstr with the added equation(s).
- 'steady_state' : A double with the same length as the number of added endogenous variables (and in the same order!). This will use the 'steady_state_fixed' option, so do not override it!

Output:

- obj : An object of class nb_dsgc.

See also:

[nb_dsgc.parse](#)

Written by Kenneth Sæterhagen Paulsen

► **addObsModelSolver** ↑

```
[A,B,C,CE,err] = nb_dsgen.addObsModelSolver(A,B,C,CE,parser,solution,...  
options,expandedOnly)
```

Description:

Expand core model with obs_model equations.

Input:

- A : Transition matrix of core model, as a nEndo x nEndo.
- C : Shock impact matrix of core model as a nEndo x nExo.
- CE : Shock impact matrix of core model as a nEndo x nExo x nHor, when solved with anticipated shocks, otherwise [].
- parser : See nb_dsgen.solveNB
- solution : See nb_dsgen.solveNB
- options : See nb_dsgen.solveNB
- expandedOnly : true if only the expanded solution is to be solved for.

Output:

- A : Transition matrix, as a nEndoAll x nEndoAll.
- B : Constant term as a nEndoAll x 1.
- C : Shock impact matrix as a nEndoAll x nExoAll - 1.
- CE : Shock impact matrix as a nEndoAll x nExoAll - 1 x nHor, when solved with anticipated shocks, otherwise [].
- ss : Steady state, as a nEndo x 1 double.

See also:

[nb_solveLinearRationalExpModel](#), [nb_dsgen.selectSolveAlgorithm](#)

Written by Kenneth Sæterhagen Paulsen

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► assignParameters ↑

obj = assignParameters(obj,varargin)

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.

- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames** ↑

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

`nb_dsge.writeTex`, `nb_dsge.writePDF`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **backwardSolver** ↑

`[A,B,BE,err] = nb_dsge.backwardSolver(model,solution,options,expandedOnly)`

Description:

Solving purly backward looking models.

Input:

- `model` : See `nb_dsge.solveNB`
- `solution` : See `nb_dsge.solveNB`
- `options` : See `nb_dsge.solveNB`
- `expandedOnly` : true if only the expanded solution is to be solved for.

Output:

- `A` : Transition matrix, as a `nEndo x nEndo`.
- `B` : Shock impact matrix as a `nEndo x nExo`.
- `BE` : Shock impact matrix as a `nEndo x nExo x nHor`, when solved with anticipated shocks, otherwise [].

Written by Kenneth Sæterhagen Paulsen

► **blockDecompose** ↑

`obj = blockDecompose(obj,inEst)`
`obj = blockDecompose(obj,inEst,doPrologue)`

Description:

Block decompose model before solving. This will remove the epilogue from the model.

Caution: This method will be called inside `nb_dsge.solve` automatically, if the 'blockDecompose' option is set to true. But you can use this method to see how efficient the block decomposition is for your model by calling it before solving.

Caution: The jacobian is evaluated at the values provided by the steady-state.

Input:

- obj : A scalar object of class nb_dsg.
- inEst : Give true if it is called inside estimation, as in this case we need to calculate the steady-state and the derivatives.
- doPrologue : Give true to also separate out prologue. Default is false.

Output:

- obj : A scalar object of class nb_dsg, where the parser property has been updated with the block decomposition of the model.

See also:

[nb_dsg](#), [nb_dsg.parse](#), [nb_dsg.solve](#), [nb_dsg.solveNB](#)

Written by Kenneth Sæterhagen Paulsen

► calculateLoss ↑

[obj, loss] = calculateLoss(obj, varargin)

Description:

Calculate loss given a specified loss function.

Input:

- obj : A N x M matrix of class nb_dsg.

Optional inputs:

- 'simulations' : An nb_ts object with the simulate data from the model.
If this option is not given, it will calculate the loss analytically.
- The rest of the optional inputs will be passed on to the nb_dsg.set.

Output:

- obj : A N x M matrix of class nb_dsg. Where the results.loss is updated.
- loss : A N x M double with the calculated loss.

See also:

[nb_dsg.setLossFunction](#)

► **calculateMultipliers** ↑

```
mult = calculateMultipliers(obj,varargin)
```

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsgc.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!
- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_dsgc.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError** ↑

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.
Extra outputs (all based on bootstrapped draws):
 - stdBeta : Standard deviation of coefficient of the main equation.
 - tStatBeta : T-statistic for the coefficient of the main equation.
 - pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
 - stdLambda : Standard deviation of coefficient of the observation equation.
 - tStatLambda : T-statistic for the coefficient of the observation equation.

- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► checkModel ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► checkPosterioriors ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they were drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot.
Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **checkSteadyState** ↑

[obj,err] = checkSteadyState(obj,varargin)

Description:

If `isNB` is true for the `nb_dsge` object `obj`, this method must be ran to check that your solution of the steady-state of the model is correct.

Use `obj = set(obj,'steady_state_file','filename.m')` to give `nb_dsge` class the file that solves the steady-state or
`obj = checkSteadyState(obj,'steady_state_file','filename.m').`

Caution: If no steady-state file is provided and the option '`'steady_state_solve'`' is false a zero for all variable solution will be tested.

Caution: If some variable is not been given any value in the steady-state file it is assumed that it is 0 in steady-state.

Input:

- `obj` : An object of class `nb_dsge`.

Optional inputs:

- `'silent'` : See `nb_dsge.help('silent')`
- `'steady_state_file'` : See `nb_dsge.help('steady_state_file')`

Output:

- `obj` : An object of class `nb_dsge`, where the solution of the steady-state can be found at `obj.solution.ss`.
- `err` : If two outputs are called for a potential error message is not thrown, but instead return as a string. Is empty if not error occurs. Caution: This is only the case if `numel(obj) == 1`.

Examples:

See `Econometrics\test\DSGE\OptimalPolicy\FRWZ\frwz_nk_nb_steadystate.m`

See also:

[nb_dsge.help](#), [nb_dsge.solveSteadyState](#)

Written by Kenneth Åkerhagen Paulsen

► **checkUpdatedPriors ↑**

`[DB,plotter,pAutocorr] = checkUpdatedPriors(obj,varargin)`

Description:

Plot updated prior draws in the order they were drawn. This to check for problems with the draws being autocorrelated.

Input:

- obj : A scalar object of class nb_model_sampling. It must represent a bayesian model that uses system priors.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_generic.checkPosteriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► cleanSolution ↑

```
obj = cleanSolution(obj)
obj = cleanSolution(obj,normalize)
```

Description:

Remove inactive shocks from solution.

Input:

- obj : A scalar nb_dsgen object.
- normalize : true or false. If true the shocks are normalized to be $N(0,1)$.

Output:

- obj : A scalar nb_dsgen object.

Written by Kenneth Sæterhagen Paulsen

► **conditionalTheoreticalMoments** ↑

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.

Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
Default is ''. See below for what that means.

> 'bootstrap' : Create artificial data to bootstrap
the estimated parameters. Default
for models estimated with classical
methods.

> 'wildBootstrap' : Create artificial data by wild
bootstrap, and then "draw" parameters
based on estimation on these data.
Only an option for models estimated

with classical methods.

> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constraints ↑**

```
[C,Ceq] = nb_model_parse.constraints(pars,constrFuncEq,constrFuncIneq,
varargin)
```

Description:

Function that will be converted into a function handle during estimation when parameter constraints are added.

Input:

- pars : The current values of the parameters.
- constrFuncEq : A function handle, which is constructed in nb_model_parse.constraints2func, that evaluate the equality constraints on the parameters.
- constrFuncIneq : A function handle, which is constructed in nb_model_parse.constraints2func, that evaluate the inequality constraints on the parameters.

Output:

- C : The value of the evaluated inequality constraints. Parameter values are feasible as long as all(C <= 0).
- Ceq : The value of the evaluated equality constraints. Parameter values are feasible as long as all(abs(Ceq) < eps), for some small eps.

See also:

[nb_dsge.constraints2func](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a

cellstr. Must be included as a variable of the data input.

- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast

```

                error.
- 'ESLS'   : Exponential of the sum of the log scores.
- 'EELS'   : Exponential of the mean of the log scores.
- 'MLS'    : Mean log score.

- allPeriods      : true if you want the scores to be calculated for all
                     periods recursively, or else give false, i.e. only
                     calculate the score for the last period. False is
                     default.

- startDate       : The date to begin constructing the score. Can be
                     empty.

- endDate         : The date to end constructing the score. Can be
                     empty.

- nSteps          : Select the number of forecasting steps. Can be
                     empty.

- rollingWindow   : Set it to a number if the combination weights are to
                     be calculated using a rolling window. Default is
                     empty, i.e. to calculate the weights recursively using
                     the full history at each recursive step.

- lambda          : Give the value of the parameter of the exponential
                     decaying weights on past forecast errors when
                     constructing the score. If empty the weights on all
                     past forecast errors are equal.

```

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **convert ↑**

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
- > third column : Any expression that can be interpreted by the nb_ts.createShift method.
- > fourth column : Comments

- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,    description
    'VAR1_G',  'pcn(VAR1)', 'avg',   'VAR1 growth'
    'VAR2_G',  'pcn(VAR2)', 'avg',   'VAR2 growth'
    'VAR3_G',  'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► curvature ↑

```
plotter = curvature(obj,beta,varargin)
```

Description:

Calculate curvature of posterior, likelihood and/or prior, wrt the estimated parameters. Can be done in parallel.

Input:

- obj : A scalar nb_dsg object.
- beta : A nPar x 1 double with the point to evaluate the curvature. If not provided the parameter values are found at obj.results.beta. Be aware that only the parameters that has been selected a prior will be analyzed! (If the model has been estimated using the estimate method, the mode will be the default point of evaluation)

Optional input:

Any combination of:

- 'prior' : Give to evaluate the curvature of the prior, at the given point.
- 'systemPrior' : Give to evaluate the curvature of the system prior, at the given point.
- 'likelihood' : Give to evaluate the curvature of the likelihood, at the given point.
- 'posterior' : Give to evaluate the curvature of the posterior, at the given point.

If non of them are provided, all of them are plotted.

Other inputs:

- 'honourBounds' : Give true to honour the lower and upper bounds of the priors when 'incrFactor' is set. Default is false.
- 'incrFactor' : Sets the width of the evaluation window. A smaller number means wider window. The increment is calculated as; incr = (ub - lb)/incrFactor, where ub is the upper bound and lb is the lower bound. Then evaluation window is calculated as; [beta - incr, beta + incr]. The number of points to evaluate in this window is set by the 'numEvalPoints'

input. E.g. curvature(obj,beta,'incrFactor',10). Default is [], i.e. use [lb,ub] as the evaluation window.

To set different increment factors for different parameters use a nPar x 1 double.

- 'numEvalPoints' : Sets the number of evaluation points inside the evaluation window. Defualt is 10. E.g. curvature(obj,beta,'numEvalPoints',50)
- 'minObjective' : Set all values less than this number to nan. Default is -1e10.
- 'parallel' : true or false. Default is false. It will run in parallel over the number of selected parameters, so number of parameters should be >> than number of selected workers to make it efficient.
- 'parameters' : A 1 x nPar cellstr with the estimated parameters of interest.
- 'subplot' : Give to create a 4 x 4 subplot. Default is not.
- 'takeLog' : A nPar x 1 logical. Set element to true if you want to take log of the bound before calculation of the evaluation window.
- 'tolerance' : If the posterior is above this number, it will be assign a nan value. Default is 1e9.
- 'waitbar' : true or false. Default is false.
- 'workers' : Give the number of workers to use during parallel. 5 is max 5, if 'waitbar' is set to true. Otherwise max is given by nb_availablePoolSize. Default is [], use the max amount of workers.

Output:

- plotter : > Default : A nPar x 1 nb_graph_data object. Use the graph method or the nb_graphMultiGUI class.
- > 'subplot' : A nb_graph_data object. Use either the graphSubPlots method or the nb_graphSubPlotGUI class.

See also:

[nb_dsge.setPrior](#), [nb_model_generic.estimate](#), [nb_dsge.curvatureEvaluator](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **declareUncertainParameters** ↑

```
obj = declareUncertainParameters(obj,varargin)
```

Description:

Use this function to declare a parameter as uncertain.

Caution: This will automatically trigger calculations of optimal simple rules under parameter uncertainty in the method nb_dsge.optimalSimpleRules.

Input:

- obj : An object of class nb_dsge.
- parameters : A N x X char or a N x 1 cellstr with the parameters you want to declare as uncertain.
- multiDist : If N == 1:
 - A scalar nb_distribution object.
- Else:
 - A nb_copula object where the distributions property hold N nb_distribution objects.

Output:

- obj : An object where the parameters property is updated, i.e. the isUncertain field and the nb_copula object is stored in the parser property.

See also:

[nb_dsge.optimalSimpleRules](#), [nb_copula](#), [nb_distribution](#)

Written by Kenneth Å!terhagen Paulsen

► **derivative ↑**

```
obj = derivative(obj,varargin)
```

Description:

Calculates the derivatives of the model with respect to the variables of the model.

Caution only first order approximation is supported for the time being.

Input:

- obj : An object of class nb_dsge.

Optional input:

```
- 'solve_order' : See nb_dsge.help('solve_order')
```

Output:

- obj : An object of class nb_dsge. See the solution property:
 - > jacobian : Storing the first-order derivatives evaluated at the steady-state of the DSGE model. The ordering will depend on the solver used.
 - > NB Toolbox: See nb_dsge.getDerivOrder

See also:

[nb_dsge.getDerivOrder](#)

Written by Kenneth Sæterhagen Paulsen

► **dieboldMarianoTest** ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two

models.

- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_dsge.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_dsge.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_dsge.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

```
obj = doSimulateFromDensity(obj,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_dsgen.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doSymbolicDerivatives** ↑

```
[derivFunc,derivInd,symDeriv,jacobian] = doSymbolicDerivatives(obj,obs)
```

Description:

Create a function handle with the derivatives.

Input:

- obj : An object of class nb_dsgen.
- obs : Give true to calculate symbolic derivatives of the obs_model instead of the core model. Default is false

Output:

- derivFunc : A function handle returning the non trivial derivatives of the equations of the model. This function takes two inputs; the value of the variables (including lead, lags and exogenous), and the value of the parameters.
- derivInd : Let JT be the output from derivFunc, then the full jacobian is given by sparse(derivInd(:,1),derivInd(:,2),JT,...

nEqs,nCol). For more see nb_dsge.derivativeNB.

- symDeriv : A vector of nb_mySD storing the non-trivial derivatives of the model.
- jacobian : The jacobian as a sparse matrix of size nEqs x nCol. See the getDerivOrder for the ordering of the jacobian.

See also:

[nb_dsge.derivative](#), [nb_dsge.derivativeNB](#), [nb_dsge.getDerivOrder](#)

Written by Kenneth SÃ¶therhagen Paulsen

► **empiricalMoments** ↑

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.

- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargin{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_dsge.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [eq](#) ↑

ret = eq(obj,other)

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density, ...
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evalLikelihood** ↑

```
lik = evalLikelihood(obj)
```

Description:

Evaluate the likelihood at the current parameters.

Input:

- obj : A scalar nb_dsge object.

Output:

- lik : Minus the log likelihood.

Written by Kenneth Sæterhagen Paulsen

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **expectedBreakPointSolver** ↑

```
[A,B,C,CE,parser,err] =  
    nb_dsgc.expectedBreakPointSolver(parser,solution,options,expandedOnly)
```

Description:

This static method of the nb_dsgc class implements the algorithm presented by; Kulish and Pagan (2017), "Estimation ans Solution of Models with Expectations and Structural Changes".

Input:

- parser : See nb_dsgc.solveNB
- solution : See nb_dsgc.solveNB
- options : See nb_dsgc.solveNB
- expandedOnly : true if only the expanded solution is to be solved for.

Output:

- A : Transition matrix, as a 1 x nStates cell array. Each element is a nEndo x nEndo double matrix.
- B : Constant term in the solution, as a 1 x nStates cell array. Each element is a nEndo x 1 double vector.
- C : Shock impact matrix, as a 1 x nStates cell array. Each element is a nEndo + nMult x nExo.
- CE : Shock impact matrix, as a 1 x nStates cell array. Each element is a nEndo x nExo x nHor, when solved with anticipated shocks, otherwise [].

Written by Kenneth Sæterhagen Paulsen

► **extendedPath** ↑

```
sim = extendedPath(obj,nSteps,varargin)
```

Description:

Perform simulation from a model using extended path.

Input:

- obj : An object of class nb_dsgc.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional input:

- 'blockDecompose' : Set it to true to use block decomposition to solve the problem.
- 'derivativeMethod' : Either 'symbolic' (fastest) or 'automatic'.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'exogenousPerm' : A cellstr with the exogenous variables of the 'exoPermVal' input.
- 'exogenous' : A cellstr with the exogenous variables of the model to simulate. If empty, all exogenous variables are simulated. All shocks are simulated from $N(0, I)$.
- 'exoPermVal' : A $N \times nExo \times draws$ double with the values of the permanent shocks to the exogenous variables used by the simulations. The value at each period is taken as the value that will be used for the entire perfect foresight of one step of the extended path simulation. If not given or given as empty, no permanent shocks will take place.

Caution: N must be equal to $nSteps$.
Caution: If this input only gives the values of a subset of the exogenous variable, the rest will be given 0 for all observations.
Caution: If this input is given, the end point of perfect foresight simulation is re-solved only when the permanent shock of this period has changed since last period. This is done before the loop of the extended path and can be done in parallel.
- 'exoVal' : A $nPer \times nExo \times nSteps \times draws$ double with the values of the exogenous variables for the simulations. If empty, the 'exogenous' input will decide which exogenous variables that is to be simulated, otherwise the 'exogenous' inputs set the name of each column of this input.

The third dimension will be equal to the number of steps to simulate, i.e. it will overrun the $nSteps$ input.
The fourth dimension will set the number of extended path simulations to be done, i.e. it will overrun the 'draws' input.

Caution: If $nPer < periods$ the values after this observation gets the values 0.
Caution: If this input only gives the values of a subset of the exogenous variable, the

rest will be given 0 for all observations.

Caution: This input must has as many pages as the nSteps input, i.e. each page is the set of anticipated shocks seen at the given period of the extended path simulation.

Caution: If this input us provided it will only give one simulation, i.e. the 'draws' is automatically set to 1.

- 'homotopyAlgorithm' : See nb_dsge.help('homotopyAlgorithm'). Default is 0, i.e. no homotopy.
- 'homotopySteps' : See nb_dsge.help('homotopySteps'). Default is 10.
- 'optimset' : A struct with the options used by the solver. See nb_perfectForesight.optimset for more.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread the simulation of the different extended paths to different threads. Default is false.
- 'periods' : The number of periods of the simulation. E.g. periods >>> number of period it takes to converge to the end values of the simulations. Default is 150.
- 'seed' : Set the seed of the pseudo random generated numbers when 'stochInitVal' is set to true. Default is 1.
- 'solver' : Name of the solver to use. Either 'nb_solve' or 'nb_abcSolve'. Default is 'nb_solve', i.e. using newton algorithm. See the 'optimset' options to adjust solving options.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

► **filter** ↑

```
obj = filter(obj,varargin)
```

Description:

Run the kalman filter on the DSGE model.

Caution : The data field of the options property must include all the observables of the model.

This method uses the RISE filter if the nb_dsge model represent a RISE object, and it uses the dynare developed filter if the underlying object is a dynare solved DSGE model (not supported in all versions of the NB toolbox). All credits should go to the authors of those codes in these cases.

Caution: Use `obj = set(obj,'data',data)` to assign the data on the observables.

Input:

- `obj` : An object of class nb_dsge

Optional Inputs:

- `varargin` : Will depend on the underlying software used:

> All:

- `'variables'` : A cellstr with the endogenous variables to store. Are you going to do shock decomposition you need all!
- `'startDate'` : Start date of filtering, as a string or a nb_date object. Default is the start date of the options.data property.
- `'endDate'` : End date of filtering, as a string or a nb_date object. Default is the end date of the options.data property.

> NBTOOLBOX:

- `'kf_init_variance'` : See `nb_dsge.help('kf_init_variance')`. Default is given by the option `'kf_init_variance'`.
- `'kf_kalmanTol'` : See `nb_dsge.help('kf_kalmanTol')`. Default is given by the option `'kf_kalmanTol'`.
- `'kf_method'` : See `nb_dsge.help('kf_method')`. Default is given by the option `'kf_method'`.
- `'kf_warning'` : See `nb_dsge.help('kf_warning')`. Default is given by the option `'kf_warning'`.
- `'smoother'` : true or false. If false only the filter is ran. Default is true.
- `'shockProps'` : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not anticipated. I.e all shocks/residuals not

provided by this input has anticipated horizon set to 1, which means that only contemporaneous shocks/residuals are seen.

> Horizon : Anticipated horizon of the shocks/residual. 1 is default.

> RISE (Must be added separately) :

- 'kf_ergodic' : Initialization at the ergodic distribution. True or false. true is default.
- 'kf_init_variance' : Initial variance factor (Harvey scale factor). If not empty, the information in T and R is ignored. Either empty or an integer.
- 'kf_user_init' : As a cell. User-defined initialization. When not empty, it can take three forms. {a0}, {a0,cov_a0}, {a0,cov_a0,PAI00} where a0 is the initial state vector with the same order as the rows of T, cov_a0 is the initial covariance of the state vector (same order as a0) and PAI00 is the initial vector of regime probabilities. Default is {}.
- 'kf_householder_chol' : If true, return the cholesky decomposition when taking the householder transformation. This option is primarily used in the switching divided difference filter. Default is false.

> Dynare (May not be supported in all version of NB toolbox) :

- 'diffuse_filter' : Either 0 or 1. See dynare documentation.
- 'lik_init' : An integer between 1 and 3. See dynare documentation.
- 'kalman_algo' : An integer between 1 and 4. See dynare documentation.
- 'Harvey_scale_factor' : See 'kf_init_variance' above.

Caution : If not provided (or empty) the default is to use the settings return while running dynare.

Output:

- obj : The results will be saved to the results property:
 - A struct with the filtering of each model stored in separate fields. Each field for each model then consist of a struct with the following fields:
 - > 'smoothed' : A struct consisting of fields of struct objects

storing the smoothed estimates.

- > 'filtered' : A struct consisting of fields of struct objects storing the filtered estimates.
- > 'updated' : A struct consisting of fields of struct objects storing the updated estimates.
- > 'likelihood' : The log likelihood. Only return when RISE is used.

See also:

[dsge.smooth](#), [dsge.filter](#), [nb_dynareSmoothe](#)

Written by Kenneth Sæterhagen Paulsen

► **findAniticipatedMatrices** ↑

B = nb_dsge.findAniticipatedMatrices(PHI,nsteps,T1AT0iT1)

Description:

Forward expansion when dealing with expected shocks.

This implements the formula find in appendiks A of Junior Maih (2010), "Conditional forecasts in DSGE models".

Let the linearized DSGE model be written as:

E_t[Theta_lead*y(t+1) + Theta_0*y(t) + Theta_lag*y(t-1) + PHI*eps(t)] = 0

And the normal solution:

y(t+1) = A*y(t) + B1*eps(t)

Input:

- B1 : Solution of the shock impact matrix, as a nEndo x nExo double. I.e.
- nAntSteps : Number of anticipated periods of the shocks.
- T1AT0iT1 : inv(Theta_lead*A + Theta_0)*Theta_lead

Written by Kenneth Sæterhagen Paulsen

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.

- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
- 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length

of the bins is given by the provided integer.

> cell : Must be on the format:

```
{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...}
```

- lowerLimit1 : An integer, can be nan, i.e. will be found.

- upperLimit1 : An integer, can be nan, i.e. will be found.

- binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given the default domain. No warning will be given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The combineForecast method of the nb_model_group class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.

- 'endDate' : The end date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is empty this input will have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).

- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.

- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.
 If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))
 If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).
- All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.
- To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.
- Caution : If empty unconditional forecast will be produced.
- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be

interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.

- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!
- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogneous variables to

condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:
 - This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

```

- 'startingProb'    : Either a double or a char:
                     > []          : If the model is filtered the
                           starting value is calculated
                           on filtered transition
                           probabilities. Otherwise the
                           ergodic mean is used.
                     > double       : A nPeriods x nRegime or a 1 x
                           nRegime double. nPeriods refer to
                           the number of recursive periods to
                           forecast.
                     > 'ergodic'    : Start from the ergodic transition
                           probabilities.

- 'stabilityTest'  : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'states'         : Either an integer with the regime to
                     forecast/simulate in, or an object of class nb_ts
                     with the regimes to condition on. The name of the
                     variable must be 'states'.

Caution: For break-point models this is decided by
          the dates of the breaks, and cannot be set
          by this option!

- 'compareToRev'   : Which revision to compare to. Default is final
                     revision, i.e. []. Give 5 to get fifth revision.
                     must be an integer. Keep in mind that this number
                     should be larger than the nSteps input.

- 'compareTo'      : Sometime you want to evaluate the forecast of one
                     variable against another variable which is not
                     part of the model. E.g. if you use the first release
                     of a variable and want to compare it against the
                     final release. To do this you can give a cellstr
                     with size n x 2, where n is the number of variables
                     to change the the actual observations to test
                     against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
                      kernel density estimation of the density
                      forecast.

- 'exoProj'        : Tolerate missing exogenous variables by projecting
                     them by a fitted model. Only when the 'condDB'
                     input is empty!

Options are:
- ''    : No projection are done. Error will be
           given if some exogenous variables are
           not given any conditional information.
           Default.

```

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.
 - Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
- Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.
- 'exoProjHist'
 - : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.
- 'exoProjDiff'
 - : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.
- 'exoProjAR'
 - : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.
 - Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjCalib'
 - : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficent 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficent 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
 - 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if

forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_dsge.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_dsge.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist ↑**

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_dsge.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► gelmanRubin ↑

```
[res,plotter] = gelmanRubin(obj)
[res,plotter] = gelmanRubin(obj,type)
```

Description:

Calculates the recursive Gelman-Rubin diagnostic of multiple M-H chains.

Input:

- obj : A scalar nb_model_sampling object.
- type : 'posterior' to test posterior draws (default) or 'updated' to test updated prior draws.

Output:

- res : A nDraws x nParam nb_data object storing the test statistic.
If any element is >> 1 run the M-H for a longer period, as it has not converged to the stationary distribution.
- plotter : A object of class nb_graph_data. Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graph on screen.

See also:

[nb_mcmc.gelmanRubinRecursive](#), [nb_dsg.e.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst, ...
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getBalancedGrowthPath** ↑

```
bgp = getBalancedGrowthPath(obj)
bgp = getBalancedGrowthPath(obj, vars, out)
```

Description:

Get steady-state solution of model as a cell matrix.

Input:

- obj : A nb_dsgc model where the steady-state solution is solved for.
- vars : The variables you want the steady-state of. Can also be (an) expression(s). Either as a cellstr or a char.
- out : Either 'cell' (default), 'double' or 'headers' (include model names as headers to the output cell, only when numel(obj)>1).

Output:

- bgp : A nEndo x (1 + nRegimes) cell matrix if vars is empty, or else the first dimension is given by length(vars) ('cell').
A nEndo x nRegimes double if vars is empty, or else the first dimension is given by length(vars) ('double').

See also:

[nb_model_generic.solution](#)

Written by Kenneth Sæterhagen Paulsen

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provide recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_dsge.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getDSGEVARPriorMoments ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by `nb_var.priorTemplate('dsge')`.

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDerivOrder** ↑

```
[order,type,ss] = getDerivOrder(obj)
```

Description:

Get order of returned jacobian.

Input:

- obj : An object of class nb_dsge.

Output:

- order : A cellstr with the order of the returned jacobian.
- type : Indicate what the derivative is take w.r.t:
 - > 1 : Leaded variables.
 - > 0 : Current variables.
 - > -1 : Lagged variables.
 - > 2 : Innovations.

```
- ss      : The steady-state of the ordered variables. As a 1 x N double.  
           If model is not solved this will return [].
```

Written by Kenneth Sæterhagen Paulsen

► **getDiagCovarianceMatrix** ↑

```
vcv = getDiagCovarianceMatrix(obj)
```

Description:

Get diagonal covariance matrix of the DSGE model. For RISE it is assumed that the shock standard deviations are parameters starting with 'std_'.

Input:

```
- obj : An object of class nb_dsg
```

Output:

```
- vcv : A nShock x 1 double
```

Written by Kenneth Sæterhagen Paulsen

► **getEndVal** ↑

```
ss = getEndVal(obj,varargin)
```

Description:

Get end values when doing permanent shocks, i.e. resolve the steady-state given some values of the exogenous variables.

The initial values that are used is the steady-state of the original model.

Input:

```
- obj : An object of class nb_dsg.
```

Optional input:

Most relevant options are;

```
- 'steady_state_block' : See nb_dsg.help('steady_state_block'). Default  
                        is to use the current value of the  
                        'steady_state_block'.
```

```
- 'steady_state_exo'      : See nb_dsgc.help('steady_state_exo'). Must be provided!
- 'homotopyAlgorithm'    : See nb_dsgc.help('homotopyAlgorithm'). Default is 0, i.e. no homotopy.
- 'homotopySteps'        : The number of homotopy steps. Default is 10.

Caution : The homotopy option 'homotopySetup' does not do anything in this method!
```

Output:

```
- ss      : A struct with the new full steady-state solution.
```

See also:

[nb_dsgc.set](#), [nb_dsgc.help](#)

Written by Kenneth Sæterhagen Paulsen

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

```
- obj   : An object of class nb_model_generic
```

Output:

```
- start : An object of class nb_date.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationTable** ↑

```
latexTable = getEstimationTable(obj)
latexTable = getEstimationTable(obj,'preamble',false)
```

Description:

Get latex table of estimated parameters.

Input:

- obj : A nb_dsg object.

% Optional input:

- 'preamble' : Give 0 if you don't want to include the preamble in the returned output. Default is to include the preamble, i.e. 1.
- 'filename' : Provide a file name to write the latex code to tex file.
- 'writePDF' : Give true (1) to write PDF using pdflatex. In this case the tex file is deleted! 'preamble' must be set to true in this case!
- 'precision' : Default is '%8.2f'. For more see second input to num2str.
- 'lookUp' : Either a N x 3 cell array, or a file that can be run by eval to create a variable lookUp that is a N x 3 cell array. The first column must list the names of the estimated parameters, the second column the latex name, and the third the description of the estimated parameter.

Caution: The latex name is enclosed in \$\$, i.e. interpreted in math mode, but description is not!

Output:

- latexTable : A one line char with the latex table code.

See also:

[num2str](#)

Written by Kenneth Sæterhagen Paulsen
Inspired by code written by Thor Andreas Aursland

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_dsge.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0, -1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is

returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!

Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_dsgc.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getHistory ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getIRFMatchingFunc ↑

```
getObjectiveFunc = getIRFMatchingFunc(obj,irfs,weights,normFunc)
```

Description:

Get objective to minimize when doing IRF matching.

Input:

- obj : An object of class nb_dsg.
- irfs : Same as the irfs output of the nb_model_generic.irf method.
- weights : Same format as the irfs input. Must provide the same shocks and variables as the irfs input, or else it must be empty (i.e. equal weights).
- normFunc : A function that maps from $R^X \rightarrow R$, where X is the number of observation of the irf to match. Default is to use $\@(\text{w})\text{norm}(\text{w})$, where w are the vector of weighted IRFs.

Output:

- getObjectiveFunc : A function handle that can be assign to the getObjectiveFunc options of the nb_dsg class.

See also:

[nb_model_generic.irf](#), [nb_dsg.estimate](#), [nb_statespaceEstimator.estimate](#)
[nb_ts](#)

Written by Kenneth Sæterhagen Paulsen

► [getLHSVars](#) ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLoss** ↑

```
loss           = getLoss(obj)
[loss,names,table] = getLoss(obj)
```

Description:

Get calculated loss from model. See nb_dsg.e.calculateLoss or nb_dsg.e.optimalSimpleRules for how to do this.

Input:

- obj : A N x M matrix of nb_dsg.e objects.

Output:

- loss : A N x M double with the losses of the different models.
- names : A N x M cellstr with the names of the models.
- table : A 2 x N*M cell with a table of the losses.

See also:

[nb_dsg.e.calculateLoss](#), [nb_dsg.e.optimalSimpleRules](#)

Written by Kenneth Sæterhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getObjective** ↑

```
[objective,beta,sigma,lb,ub] = getObjective(obj)
```

Description:

Get objective for doing sampling from the posterior distribution of an estimated DSGE model. This is what is being minimized.

Input:

- obj : An object of class nb_dsg.

Output:

- objective : A function handle calculating the objective function that where maximized.
- beta : A nParam x 1 double with the values of all the parameters of the model.
- sigma : Covariance matrix of the estimated parameters at the mode.
- lb : Lower bound of the support of the estimated parameters, as a nEst x 1 double.
- ub : Upper bound of the support of the estimated parameters, as a nEst x 1 double.

See also:

Written by Kenneth Sæterhagen Paulsen

► **getObjectiveForEstimation** ↑

```
[fh,estStruct,lb,ub,opt,options] = ...
    nb_dsgen.getObjectiveForEstimation(options,forSystemPrior)
```

Description:

Get objective to minimize. Used by the nb_statespaceEstimator.estimate function.

See also:

[nb_dsgen.estimate](#), [nb_statespaceEstimator.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **getOptimalMonetaryPolicyMatrices** ↑

```
[Hlead,H0,Hlag,D] = ...
    nb_dsgen.getOptimalMonetaryPolicyMatrices(Alead,A0,Alag,B,W,beta)
```

Description:

Form the optimal monetary policy matrices when the Klein algorithm is used to solve the problem.

See also:

[nb_dsgen.optimalMonetaryPolicySolver](#)

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit          = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will
get the PIT from the start date of the recursive forecast and
use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct
the PITs based on the period from the estimation start date
until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a
nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date

cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

: Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions** ↑

```
distr = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.

- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPriorDistributions** ↑

```
[distr,paramNames] = getPriorDistributions(obj)
```

Description:

Get prior distributions of model.

Input:

- obj : An object of class nb_dsgc.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

► **getRecursiveEstimationGraph ↑**

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

residualNames = getResidualNames(obj)

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

roots = getRoots(obj)

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursivly estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
 - type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- If the object input is a vector, the type input can also be a cellstr array with matching length.
- Caution: See comment to the invert input!
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
 - startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
 - endDate : The date to end constructiong the score. Can be

empty. Either as a string or an object of class nb_date.

- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getSteadyState** ↑

```
ss = getSteadyState(obj)
ss = getSteadyState(obj,vars,out)
```

Description:

Get steady-state solution of model as a cell matrix.

Input:

- obj : A nb_dsge model where the steady-state solution is solved for.
- vars : The variables you want the steady-state of. Can also be (an) expression(s). Either as a cellstr or a char.
- out : Either 'cell' (default), 'struct', 'double' or 'headers' (include model names as headers to the output cell, only when numel(obj)>1).

Output:

- ss : A nEndo x (1 + nRegimes) cell matrix if vars is empty, or else the first dimension is given by length(vars) ('cell').
A nEndo x nRegimes double if vars is empty, or else the first dimension is given by length(vars) ('double').

See also:

[nb_model_generic.solution](#)

Written by Kenneth Åkerhagen Paulsen

► **getUpdatedPriorDistributions** ↑

```
distr          = getUpdatedPriorDistributions(obj)
[distr,paramNames] = getUpdatedPriorDistributions(obj,'draws',1000)
```

Description:

Get the updated prior distributions of model. Only from the first chain.

Input:

- obj : An object of class nb_model_sampling.

Optional input:

- 'draws' : The number of draws to sample from the updated prior to base the kernel estimation on, if empty or not provided all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► getVariablesList ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► geweke ↑

```
res = geweke(obj)
res = geweke(obj,chain,type)
```

Description:

Geweke (1992) test. This test split sample into two parts. The first 10% and the last 50%. If the chain is at stationarity, the means of two samples should be equal. The null hypothesis is that the subsamples has the same mean.

Input:

- obj : A scalar nb_model_sampling object.
- chain : Give the chain to test. If [] all chains are tested (default).
- type : 'posterior' to test posterior draws (default) or 'updated' to test updated prior draws.

Output:

- res : A 2 x nParam x nChains nb_cs with the difference in mean statistics ('statistic') and p-values ('P-value').

See also:

[nb_mcmc.geweke](#), [nb_dsge.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► graphCorrelation ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against emprirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- `plotter` : An object of class `nb_graph_cs`. Use the `graph` method or the `nb_graphPagesGUI` class.

See also:

`nb_dsg.e.simulatedMoments`, `nb_dsg.e.theoreticalMoments`
`nb_dsg.e.empiricalMoments`, `nb_graphPagesGUI`, `nb_graph_cs`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a `nb_model_estimate` object handle conditional info.

Input:

- `obj` : A scalar `nb_model_etimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a `nb_model_estimate` object handle missing observations.

Input:

- `obj` : A scalar `nb_model_etimate` object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_dsge.help
helpText = nb_dsge.help(option)
helpText = nb_dsge.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you

need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.

- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_dsgc.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretStochasticTrendInit** ↑

```
stInit = nb_dsgc.interpretStochasticTrendInit(parser, ...
                                               stochasticTrendInit, beta)
```

Description:

Interpret the stochasticTrendInit options.

Input:

- parser : A struct. See the property nb_dsge.parser.
- stochasticTrendInit : Either a struct or function_handle. See the examples in the folder Examples\Econometrics\... DSGE\StochasticTrend for both options.
- beta : A nParam x 1 double with the parameter values. In the case that the stochasticTrendInit input is a function handle these are passed as a struct, where the fieldnames are the parameter names and the field are their values.
- parameters : A nParam x 1 cellstr with the parameter names.

Output:

- stInit : A nEndo x 1 double with the initial condition. Useful for models with level variables.

Written by Kenneth Sæterhagen Paulsen

► irf ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsge objects.
- 'compare' : Give true if you have a scalar nb_dsge model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

```

> 'cumulative product' : cumprod(1 + X,1)
> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
      the map from structural shocks to dependent
      variables. See nb_var.set_identification. Of
      course this option only makes sense for VARs
      identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

```

- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.
- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

```

- 'startingProb'      : Either a double or a char (Only for Markov-switching
models):

    > scalar           : Select the the regime to take the
                           initial transition probabilities
                           from.

    > double            : A draws x nRegime or a 1 x
                           nRegime double. draws refer to
                           the number set by the 'draws'
                           input.

    > 'ergodic'         : Start from the ergodic transition
                           probabilities. Default for 'irf'.

    > 'random'          : Randomize the starting values
                           using the simulated starting
                           points. Default for 'girf'.

- 'startingValues'   : Either a double or a char:

    > double            : A nVar x 1 double.

    > 'steadystate'     : Start from the steady state. If you are dealing
                           with a MS-model or a break-point model you can
                           indicate the regime by 'steadystate(regime)'. E.g.
                           to start in regime 1 give; 'steadystate(1)'. For
                           MS - models the default is the ergodic mean
                           (default as long as 'girf' is not selected as
                           'type'), while for break-point models it is to
                           start from the first regime.

    > 'zero'             : Start from zero on all variables.

    > 'random'           : Randomize the starting values using the simulated
                           starting points. Default when 'type' set to
                           'girf'.

- 'states'            : Either an integer with the states to produce irf in,
                           or a double with the states to condition on. Must
                           have size periods x 1 in the last case. Applies to
                           both MS - models and break-point models.

- 'type'               : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf'             : Generlized impulse response function
                           calculated as the mean difference between
                           shocking the model with a one period shock
                           (1 std) and no shock at all. Use the 'draws'
                           input to set the number of simulations to use
                           to base the calculation on. This type of
                           irfs must be used for non-linear models.

    > 'irf'                : Standard irf for linear models. Calculated by
                           shocking the model with a one period shock
                           (1 std).

- 'variables'          : The variables to create impulse responses of.

```

Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_dsge.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isBreakPoint** ↑

```
ret = isBreakPoint(obj)
```

Description:

Is the nb_dsge object using the NB Toolbox solver for DSGE models with unanticipated break points or not.

Input:

- obj : A nb_dsge object.

Output:

- ret : 1 if true, else 0.

Written by Kenneth Sæterhagen Paulsen

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsge object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsg objects!

For nb_dsg objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isRise** ↑

```
ret = isRise(obj)
```

Description:

Is the nb_dsg object a wrapper of a RISE dsg object or not.

Input:

- obj : A nb_dsg object

Output:

- ret : 1 if true, else 0.

Written by Kenneth SÃ¶terhagen Paulsen

► **isSampled** ↑

```
ret = isSampled(obj)
```

Description:

Is the nb_model_sampling object sampled or not.

Input:

- obj : A nb_model_sampling object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isStationary** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

ret = isestimated(obj)

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint

```
prediction bands for
macroeconomic risk management

> 'copula' : Using a copula likelihood approach.

> 'mostlik' : Group the paths based on how
likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the
error bands. If empty (default) all forecasting steps stored
in the object is used.
```

Output:

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.
- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **kleinSolver** ↑

```
[A,D,err] = nb_dsge.kleinSolver(parser,FF,F0,FB,FU)
```

Description:

Uses the Klein algorithm to solve the rational expectation mode.

See also:

[nb_solveLinearRationalExpModel](#)

Written by Kenneth Sætherhagen Paulsen

► **likelihood** ↑

```
[fval,sol] = nb_dsge.likelihood(par,estStruct)
```

Description:

Evaluate minus the log likelihood.

Input:

- See the method `nb_model_generic.objective` for description of inputs.

Output:

- `fval` : Value of minus the log likelihood at the given parameters.
- `sol` : A struct with the solution of the model. See output from `nb_dsg.e.stateSpace`,

See also:

`nb_dsg.e.objective`, `nb_dsg.e.stateSpace`, `nb_dsg.e.stateSpaceBreakPoint`
`nb_dsg.e.stateSpaceTVP`, `nb_kalmanLikelihoodMissingDSGE`
`nb_kalmanLikelihoodBreakPointDSGE`, `nb_kalmanLikelihoodTVPDSGE`
`nb_kalmanLikelihoodUnivariateStochasticTrendDSGE`

Written by Kenneth Sæterhagen Paulsen

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- `obj` : An object of class `nb_model_generic`.

Output:

- `posterior` : A struct with the posterior output (The format will depend on the sampling method used).

See also:

`nb_model_estimate.estimate`, `nb_dsg.e.assignPosteriorDraws`
`nb_model_sampling.sample`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **looseOptimalMonetaryPolicySolver ↑**

```
[H,D,DE,parser,err] = ...
    nb_dsge.looseOptimalMonetaryPolicySolver(parser,solution, ...
        options,expandedOnly)
```

Description:

This static method of the nb_dsge class implements the algorithm presented by; Debortoli, Maih and Nunes (2010) "Loose commitment in medium-scale macroeconomic models: Theory and an application".

All credits should go to the authors of that paper for the solution and not the writer of this code.

Input:

- parser : See nb_dsge.solveNB
- solution : See nb_dsge.solveNB
- options : See nb_dsge.solveNB
- expandedOnly : true if only the expanded solution is to be solved for.

Output:

- H : Transition matrix, as a nEndo + nMult x nEndo + nMult.
- D : Shock impact matrix as a nEndo + nMult x nExo.
- DE : Shock impact matrix as a nEndo x nExo x nHor, when solved with anticipated shocks, otherwise [].

Written by Kenneth Sæterhagen Paulsen

► **makeObservationEq ↑**

```
H = nb_dsge.makeObservationEq(obsInd,nEndo)
```

Description:

Make observation equation of NB toolbox solved DSGE model.

Input:

- obsInd : A vector of integers with length nObs storing the location of the observables in the vector of endogenous variables.
- nEndo : The number of endogenous variables.

Output:

- H : A nObs x nEndo matrix.

Written by Kenneth Sæterhagen Paulsen

► mcfRestriction ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. @(x,y)gt(x,y),
@(x,y)gt(x-y,0)||lt(x-y,1), i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:
 1. The expression using any of the endogenous variables of the model.
 2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_dsgen.monteCarloFiltering](#), [nb_dsgen.irf](#)
[nb_dsgen.theoreticalMoments](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [meanPlot](#) ↑

```
[res,plotter] = meanPlot(obj)
[res,plotter] = meanPlot(obj,periods,type)
```

Description:

Calculates the (moving) mean of each sampled chain and plot it.

Input:

- obj : A scalar nb_model_sampling object.
- periods : Maximum period length when calculating the moving mean.
 Default is to use the full sample.
- type : 'posterior' to test posterior draws (default) or 'updated' to test updated prior draws.

Output:

- res : A nDraws x nParam x nChain nb_data object with the (moving) mean of the parameters.
- plotter : A object of class nb_graph_data. Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graph on screen.

See also:

[nb_dsge.sample](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► mincerZarnowitzTest ↑

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_dsge.uncondForecast](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► monteCarloFiltering ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_dsg.e.mcfRestriction](#), [nb_dsg.e.solve](#)
[nb_dsg.e.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► nb2RisePreparser ↑

```
newFile = nb_dsg.e.nb2RisePreparser(rise_file,silent,macroProcessor,...  
macroVars)
```

Description:

This static method may be useful to convert a nb model file to a rise model file.

Caution: This is provided with no guarantee it will work, and a lot of the special options in a nb model file is not supported in RISE, and these options are not checked by this method!

Input:

- rise_file : A .mod or .nb file with the model to run in RISE instead of using the NB toolbox parser and solver.
- silent : If the parsing should be silent or not.
- macroProcessor : Run NB macroprocessor before writing it to a .rs file.
- macroVars : A vector of nb_macro objects. Use for the macroprocessing. Default is an empty nb_macro array.

Output:

- newFile : A .rs file with the file you can run with RISE.

See also:

[nb_dsg](#)

Written by Kenneth Sæterhagen Paulsen

► **objective** ↑

```
fval = nb_dsg.objective(par,estStruct)
```

Description:

The objective to minimize when doing estimation

Input:

```
- par : Current vector of the estimated parameters.  
- estStruct : A struct with at least the following fields;  
    > indPar : Index for where in the full parameter vector to find  
        the estimated parameters.  
    > beta : A vector of the values of all parameters. As a  
        nParam x 1.  
    > model : The obj.parser property of the nb_dsg class.  
    > options : The obj.estOptions property of the nb_dsg class.  
        (After calling getEstimationOptions, i.e. calling  
        the estimate method.)  
    > y : A nObservables x nPeriods double with the data to  
        estimate the model on. The data may contain missing  
        observations.  
    > z : Not in use. To make it generic to other classes of  
        models.  
    > observables : A logical vector. An element is true if the variable  
        is observable.
```

Output:

```
- fval : Value of the objective at the given parameters.
```

See also:

[nb_model_generic.estimate](#), [nb_dsg.likelihood](#)

Written by Kenneth Sæterhagen Paulsen

► **optimalMonetaryPolicySolver** ↑

```
[H,D,DE,parser,err] = ...
nb_dsge.optimalMonetaryPolicySolver(parser,solution,options, ...
expandedOnly,solveForEpi)
```

Description:

This static method of the nb_dsge class implements the Klein algorithm to solve the problem.

Caution: This algorithm can only solve optimal monetary policy under full commitment or full discretion.

Input:

- parser : See nb_dsge.solveNB
- solution : See nb_dsge.solveNB
- options : See nb_dsge.solveNB
- expandedOnly : true if only the expanded solution is to be solved for.

Output:

- H : Transition matrix, as a nEndo + nMult x nEndo + nMult.
- D : Shock impact matrix as a nEndo + nMult x nExo.
- DE : Shock impact matrix as a nEndo x nExo x nHor, when solved with anticipated shocks, otherwise [].

Written by Kenneth Sæterhagen Paulsen

► **optimalSimpleRules** ↑

```
obj = optimalSimpleRules(obj,simpleRules)
obj = optimalSimpleRules(obj,simpleRules,varargin)
[obj,loss,osr_coeff,std_osr_coeff] = ...
optimalSimpleRules(obj,simpleRules,varargin)
```

Description:

Compute optimal simple rules given a loss function. The loss function can either be specified in the model file or by the nb_dsge.setLossFunction.

To set the discount factor of the optimizing authorities set the 'lc_discount' option. E.g. obj = set(obj,'lc_discount',0.99). This option can also be set in the model file, for more on this see DAG.pdf.

Caution: The return object is solved with the optimized simple rules.

Input:

- obj : An object of class nb_dsg.e.
- simpleRules : A char or cellstr array with the rules to maximize.
Each rule must have the instrument as its only left hand variable. E.g. ' $i = lam1*pi + lam2*y$ ' or
 $\{i = lam1*pi + lam2*y; tau = lam3*pi + lam4*y\}$. The lamX parameters are in this case the parameters that is optimized over.
- type : Either 'commitment' or 'discretion'. If 'discretion' the interest rate rule can only respond to state variables! Default is 'commitment'.

Optional inputs:

- 'init' : Initial values of the optimization routine. As a struct with the fieldnames as the parameter names of the simple rules and the fields as the initial values.

Caution: If the parameters are declared in the model file (.nb file) you may use nb_dsg.e.assignParameters before calling this function instead, i.e. the calibration is used as the initial values.
- 'lowerBound' : Sets the lower bound on the coefficients of the simple rules.
- 'osr_type' : Either 'commitment' or 'discretion'. If 'discretion' the interest rate rule can only respond to state variables! Default is 'commitment'.
- 'upperBound' : Sets the upper bound on the coefficients of the simple rules.

Else:

- varargin : See the the set method.

Output:

- obj : An object of class nb_dsg.e. The model is solved with the optimized simple rule.

Examples:

```
obj = optimalSimpleRules(obj,'i = lam1*pi + lam2*y');
```

See also:

[nb_dsg.e](#), [nb_dsg.e.parse](#), [nb_dsg.e.setLossFunction](#)

Written by Kenneth Sæterhagen Paulsen

► parameterDraws ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated

with classical methods.

```

> 'wildBlockBootstrap' : Create artificial data by wild
  overlapping random block length
  bootstrap., and then "draw" parameters
  based on estimation on these data.
  Only an option for models estimated
  with classical methods.

> 'copulaBootstrap'      : Uses a copula approach to draw residual
  that are autocorrelated, Does not handle
  heteroscedasticity. Only an option for
  models estimated with classical methods.

> 'posterior'           : Posterior draws. Only for models
  estimated with bayesian methods.
  Default for models estimated with
  bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

```

- 'parallel'      : Run in parallel. true or false.

- 'cores'         : Number of cores to use. Default is to use all cores
  available.

- 'newDraws'       : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the draws input.

  Caution: When setting 'parallel' to true on MATLAB
  version later than R2017B this number will
  be the number of new draws, and not the
  factor!!! E.g. set it to 100. If it is given
  as a number less than 1, it will by default
  be set to 100.

- 'initialDraws'  : When drawing parameters this factor decides how many
  many draws that are produced before solving the model.
  Default is 1. I.e. it is equal to draws input. For
  VAR models with underidentification it may be a good
  idea to set this to a value > 1 as you may drop a
  lot of parameters when identification fails. Default
  is 1.

```

Output:

```

- out          : The output depend on the input output:

> 'param'      : Default. A struct with the following fields:

  beta   : A nPar x nEq x draws double with the

```

estimated parameters.

sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

ci = parameterIntervals(obj,alpha)

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

```
- ci      : A nPar x 3 cell matrix.  
Written by Kenneth SÃ¶terhagen Paulsen  
Inherited from superclass NB_MODEL_GENERIC
```

► **parse** ↑

```
obj = nb_dsgc.parse(filename,varargin)
```

Description:

Parse model file on the .nb format.

Input:

- filename : A string with the name of the model file with extension .nb.

Optional inputs:

(Most relevant to parsing)

- 'macroProcessor' : See nb_dsgc.help('macroProcessor').
- 'macroVars' : See nb_dsgc.help('macroVars').
- 'silent' : See nb_dsgc.help('silent').

Output:

- obj : An object of class nb_dsgc.

See also:

[nb_dsgc.solve](#), [nb_dsgc.help](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **parseTempParameters** ↑

```
out = nb_model_parse.parseTempParameters(eqs,verbose)
```

Description:

Parse terms starting with #, and substitue out for where these terms are used.

Input:

- eqs : A N x 1 cellstr with the equations of the model.
- verbose : Set to true to print resulting equations.

Output:

- out : A N x 1 cellstr with the parsed equations of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **perfectForesight** ↑

```
[sim,plotter,distr] = perfectForesight(obj,varargin)
```

Description:

Perform perfect foresight simulation.

Input:

- obj : An object of class nb_dsg.

Optional input:

- 'addEndVal' : Add end value as the last observation of the simulation results, i.e. if the 'periods' input is set to 100, the sim output will return a nb_data object with 101 observation. nan is given to the variable not assign a end value with the 'endVal' input. Default is true.
- 'addSS' : Set to false to not add the steady-state to the simulation. Default is true.
- 'blockDecompose' : Set it to true to use block decomposition to solve the problem.
- 'derivativeMethod' : Either 'symbolic' (fastest) or 'automatic'.
- 'draws' : Number of draws from the distribution of the initial values. Default is 500. See 'stochInitVal'.
- 'endVal' : Sets the end values of the simulation of all the endogenous variables of the model. Must be given as a struct where the fieldnames are the names of the endogenous variables of the model, and the fields are their end value. If not provided the end values will be the original steady-state. See the nb_dsg.getEndVal method to find the end values after a permanent shock.

- 'exoVal' : A nb_data object with the values of the exogenous variable for the wanted simulation.
 - Caution: If startObs > 1, the values before this observation gets the values 0.
 - Caution: If endObs < periods the values after this observation gets the values 0.
 - Caution: If this input only gives the values of a subset of the exogenous variable, the rest will be given 0 for all observations.
- 'homotopyAlgorithm' : See nb_dsge.help('homotopyAlgorithm'). Default is 0, i.e. no homotopy.
- 'homotopySteps' : See nb_dsge.help('homotopySteps'). Default is 10.
- 'initVal' : Sets the initial values of the simulation of all the backward looking variables. Must be given as a struct where the fieldnames are the names of the backward looking variables of the model, and the fields are their initial value. If not provided the initial values will be the original steady-state.
 - Caution: If 'stochInitVal' is set to true, this option will set the conditional information of the initial condition. For more on this case see doc on the input 'stochInitVal'.
- 'optimset' : A struct with the options used by the solver. See nb_perfectForesight.optimset for more.
- 'periods' : The number of periods of the simulation. E.g. periods >>> number of period it takes to converge to the end values of the simulations. Default is 150.
- 'plotInitSS' : Plot initial steady-state in the return graphs, if 'addSS' is set to false, zero lines will be plotted. The period 0 will also be included, and will be set to the initial steady-state (zero). true or false. Default is false.
- 'plotSS' : Plot steady-state values during simulation. 'plotInitSS' will be set to true in this case.
- 'seed' : Set the seed of the pseudo random generated numbers when 'stochInitVal' is set to true. Default is 1.
- 'solver' : Name of the solver to use. Either 'nb_solve' or 'nb_abcSolve'. Default is 'nb_solve', i.e. using newton algorithm. See the 'optimset' options to adjust solving options.

- 'startingValues' : How to find the starting values of the problem to solve, i.e. the initial values of the full path.
 - > 'steady_state' : All variables for all periods are set to their initial steady-state values. Default.
- 'stochInitVal' : Set to true to draw initial conditions. If 'initVal' is not provided it will draw from unconditional distribution of the initial condition, or else it will draw from the conditional distribution of the initial values. In the latter case you can fix the value of a variable by giving a scalar double to a field of the 'initVal' input, or you can specify a truncated interval by providing a 1x2 double with the lower and upper bound to a field of the 'initVal' input. E.g:
 1. Condition on a point: init.Var1 = 2;
 2. Condition on an interval: init.Var1 = [1.5,2.5];

Output:

- sim : The perfect foresight simulation stored as a periods x nVar nb_data object
- plotter : A nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the figures.

See also:

[nb_model_generic.irf](#), [nb_dsge.getEndVal](#), [nb_dsge.checkSteadyState](#)

Written by Kenneth Sæterhagen Paulsen

► **permanentShock ↑**

[sim,plotter,ssTable] = permanentShock(obj,varargin)

Description:

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'algo' : Either 'breakPoint' or 'perfectForesight', default is 'perfectForesight'.
- 'delta' : Give true, if you want to run the delta on the

shock hitting, not hitting. This can be utilized to explore the differences in IRFs from shocks when initial values are different from the initial steady-state. I.e. when using the 'initVal' input. Default is false.

Caution: This option is not accepted for 'algo' set to 'breakPoint'.

- 'periods' : See the 'periods' input to the perfectForesight method.
- 'steady_state_block' : See nb_dsge.help('steady_state_block'). Default is to use the current value of the 'steady_state_block'. Default is true.
- 'steady_state_exo' : The exogenous innovations that are subject to a permanent shock. See nb_dsge.help('steady_state_exo') for the format of this input. Must be provided!
- All the inputs to the nb_dsge.perfectForesight method except; 'endVal', 'exoVal' and 'periods'. Inputs like 'initVal' and 'stochInitVal' will have no impact when 'algo' is set to 'breakPoint'.

Output:

- sim : See the doc of the same output from the nb_dsge.perfectForesight method.
- plotter : See the doc of the same output from the nb_dsge.perfectForesight method.
- ssTable : A nEndo x 3 nb_cs object with the old steady state, new steady state and the difference.

See also:

[nb_dsge.getEndVal](#), [nb_dsge.perfectForesight](#), [nb_dsge.checkSteadyState](#)
[nb_dsge.addBreakPoint](#)

Written by Kenneth Sæterhagen Paulsen

► **plotForecast ↑**

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF ↑**

plotter = nb_model_generic.plotMCF(paramD, params, lowerBound, ...
upperBound, method)

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- `plotter` : > 'allInOne' : A `nb_graph_cs` object. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
- > 'biplot' : A vector of `nb_graph_data` objects with size equal to the number of pairwise combination of the parameters that can be made. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_dsgc.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► `plotMCFDistTest` ↑

`plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)`

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `test` : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_dsgc.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_dsgc.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_dsgc.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPriors](#) ↑

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotUpdatedPriors** ↑

```
plotter = plotUpdatedPriors(obj)
plotter = plotUpdatedPriors(obj,varargin)
```

Description:

Plot updated priors (and priors) drawn during estimation. Only for bayesian models. Only from the first chain.

Input:

- obj : A scalar object of class nb_model_sampling. It must represent a bayesian model that is estimated using system priors. See nb_model_sampling.sampleSystemPrior.

Optitonal input:

- 'prior' : Give this string as an input to compare updated priors with the prior distribution (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the updated prior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- `plotter` : A 1 x numCoeff vector of objects of class `nb_graph_data`. Use either the `graph` method or the `nb_graphMultiGUI` class to plot the updated priors on screen.

Caution: If '`subplot`' is given it will return a scalar `nb_graph_data` object. Use the `graphSubPlots` method or the `nb_graphSubPlotGUI` class to plot the updated priors on screen.

See also:

`nb_graph_data`, `nb_graphMultiGUI`, `nb_graphSubPlotGUI`
`nb_dsgc.getUpdatedPriorDistributions`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_SAMPLING`

► **print** ↑

`printed = print(obj)`

Description:

Get the estimation result printed to a char array.

Input:

- `obj` : A vector of `nb_model_estimate` objects.

Output:

- `printed` : A char array with the estimation results.

See also:

`nb_model_estimate.print_estimation_results`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **printCov** ↑

`printed = printCov(obj)`

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► printDecisionRules ↑

```
str = printDecisionRules(obj)
str = printDecisionRules(obj,variables)
str = printDecisionRules(obj,variables,precision,asCell)
```

Description:

Print decision rules.

If the model is solved by RISE this calls the dsge.print_solution method made by Junior Maih, which is part of the RISE toolbox.

Input:

- obj : An object of class nb_dsge.
- variables : A cellstr with the variables to print the solution for.
- precision : The precision of the printed decision rules.
- asCell : Give true to return decion rules table as cell. Not supported if numel(obj) > 1 && (length(variables) > 1 || length(variables) == 0). Default is false.

Output:

- str : A char with the printed decision rules of the model. If numel(obj) > 1 the output will be a 1 x nobj cell array.

If asCell is set to true it will be a cell array instead.

See also:

[nb_dsge.solve](#), [nb_dsge.findRESolution](#), [dsge.print_solution](#)

Written by Kenneth Sæterhagen Paulsen

► **printOSR** ↑

```
res = nb_dsgc.printOSR(results,options,precision)
```

Description:

Get the results od the optimization of the simple rules as a char.

Input:

- results : A struct with the results from the optimization of the simple rules.
- options : The options property of the nb_dsgc object.
- precision : The precision of the printed result.

Output:

- results : A char with the estimation results.

Written by Kenneth Sæterhagen Paulsen

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_dsgc.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **priorPredictiveAnalysis** ↑

```
varargout = priorPredictiveAnalysis(obj,method,draws,rmsd,varargin)
```

Description:

Draw from the independent prior distributions and get the prior predictive distribution of the object of interest.

Input:

- obj : An object of class nb_dsge.
- method : The method of study. A string with the name of the method. One of; 'irf','calculateMultipliers', 'simulatedMoments' or 'theoreticalMoments'.
- draws : The number of draws made from the independent priors.
- rmsd : Calculate RMSD. See more under the documentation of the outputs.

Optional input:

- The inputs that are passed to the specified method, i.e.
varargout = method(obj,varargin). See also the examples.

Output:

- The optional outputs return by the specified method, i.e.
varargout = method(obj,varargin). See also the examples.

Caution: If the rmsd input is set to true all the outputs are returned as structures. One field called the 'prior' contain the prior predictive analysis when all parameters are varied. The fields that have the names of a parameter, we have kept that parameter at its mean (Be aware that some simulation may fail in this case and that the simulation in this case may be lower than the draws input). The corresponding RMSD is reported in in the <paramName>RMSD. This only applies to output that are nb_ts, nb_data, nb_cs or a struct of the already mentioned classes.

Examples:

```
[irfs,~,plotter] = priorPredictiveAnalysis(obj,'irf',1000,,false,...  
    'shocks',{'E_X'},'variables',{'X','Y'},...)  
  
mult = priorPredictiveAnalysis(obj,'calculateMultipliers',1000,true,...  
    'variables',{'Y','Z'},'instrument','X',...  
    'rate','R','shocks',{'E_X'})
```

See also:

[nb_model_generic.irf](#), [nb_model_generic.calculateMultipliers](#)

Written by Kenneth Sætherhagen Paulsen

► **rationalExpectationSolver** ↑

```
[A,D,DE,err] = nb_dsge.rationalExpectationSolver(parser,solution,...  
options)
```

Description:

Solving forward looking models under the rational expectation assumption.

This uses the solution approach put forward by Paul Klein. See
nb_solveLinearRationalExpModel.

Input:

- parser : See nb_dsge.solveNB
- solution : See nb_dsge.solveNB
- options : See nb_dsge.solveNB
- expandedOnly : true if only the expanded solution is to be solved for.

Output:

- A : Transition matrix, as a nEndo x nEndo.
- D : Shock impact matrix as a nEndo x nExo.
- DE : Shock impact matrix as a nEndo x nExo x nHor, when solved with anticipated shocks, otherwise [].

See also:

[nb_solveLinearRationalExpModel](#), [nb_dsge.selectSolveAlgorithm](#)

Written by Kenneth Sæterhagen Paulsen

► **removeBreakPoints** ↑

```
obj = removeBreakPoints(obj)
```

Description:

Remove all break-points added to the structural parameters of the model.

Input:

- obj : An object of class nb_dsge.

Output:

- obj : An object of class nb_dsg.

See also:

[nb_dsg.addBreakPoint](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations** ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **reparse** ↑

obj = reparse(obj,varargin)

Description:

Re-parse model given new parsing options.

Input:

- obj : An object of class nb_dsg.

Optional input:

(Most relevant to parsing)

- 'macroVars' : See nb_dsge.help('macroVars').
- 'silent' : See nb_dsge.help('silent').

Output:

- obj : An object of class nb_dsge.

See also:

[nb_dsge.set](#), [nb_dsge.parse](#)

Written by Kenneth Å!terhagen Paulsen

► **reportFiltered** ↑

```
[reported,raw] = reportFiltered(obj,expression)
[reported,raw] = reportFiltered(obj,expression,extra)
[reported,raw] = reportFiltered(obj,expression,extra,type,normalize, ...
    econometricians)
```

Description:

Input:

- obj : An object of class nb_dsge.
- expression : See the documentation of the reporting property of the nb_dsge class.
- extra : An object of class nb_ts with additional variables used for reporting purposes.
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Output:

- reported : An object of class nb_ts with the reported variables.
- raw : An object of class nb_ts with the raw filtered variables.

Written by Kenneth Å!terhagen Paulsen

► **sample** ↑

```
obj = sample(obj,varargin)
```

Description:

Sample from the posterior distribution using some type of sampling algorithm.

Caution: Be aware that the 'draws' options only decides the number of draws used to produce error bands of IRF, forecast etc., while the the 'draws' field of the 'sampler_options' options sets the number of the sampler itself. The draws to keep are randomly selected out of the sampled draws!

Caution: If you use an external sampler, you can use the nb_model_generic.assignPosteriorDraws method to assign this to the object. Be caution to not assign too many draws, as you may run out of memory (i.e. don't assing 1 000 000 draws)!

Caution: To manually inspect the posterior draws use nb_model_generic.loadPosterior, and see the field output. This is the output from the sampler function specified.

Input:

- obj : An object of class nb_model_sampling.

Optional input:

- varargin : Optional input pairs given to the set method. See nb_model_estimate.set. I.e. these input can be used to set fields of the options property. Most relevant is 'sampler_options'

Output:

- obj : An object of class nb_model_sampling.

See also:

[nb_model_estimate](#), [nb_model_generic.assignPosteriorDraws](#)
[nb_model_generic.checkPosteriors](#), [nb_model_generic.plotPosteriors](#)
[nb_dsge.geweke](#), [nb_dsge.gelmanRubin](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► **sampleSystemPrior** ↑

```
obj = sampleSystemPrior(obj,varargin)
```

Description:

Sample from the system prior distribution using some type of sampling algorithm, i.e. get the prior distribution of the parameters given the system priors.

Input:

- obj : An object of class nb_model_sampling.

Optional input:

- varargin : Optional input pairs given to the set method. See nb_model_estimate.set. I.e. these input can be used to set fields of the options property. Most relevant is 'sample_options'

Output:

- obj : An object of class nb_model_sampling.

See also:

[nb_model_estimate](#), [nb_model_generic.assignPosteriorDraws](#)
[nb_dsgc.systemPriorObjective](#)

Written by Kenneth Åstrehagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► [set ↑](#)

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setLossFunction ↑**

obj = setLossFunction(obj, lossFunction, optimal)

Description:

Set the loss function to optimize, be it with optimal simple rules, with full optimal monetary policy solution or for calculating the loss using the calculateLoss method.

Caution: If you already have solved the model under optimal policy, you can reset the loss function with the optimal input set to true, and then call the solve method to re-solve the model.

Tip : If you already have solved the model under optimal policy, and you for some reason want to calculate the loss with another loss function than in the model, you can use this method with the optimal input set to true. Do not call the solve method in this case!

Input:

- obj : An object of class nb_dsgen.

- lossFunction : A string with the loss function. E.g.
`'0.5*(INF2+LAM_Y*Y2)'.`
- optimal : Set it to false to not trigger solving the model under optimal monetary policy, i.e. this loss function can then be used to calculate the loss of a DSGE solved with taylor type rule. Default is true.

Output:

- obj : An object of class nb_dsge, where the parser property has been updated with the new loss function.

See also:

`nb_dsge.solve`, `nb_dsge.optimalSimpleRules`
`nb_dsge.looseOptimalMonetaryPolicySolver`
`nb_dsge.optimalMonetaryPolicySolver`, `nb_dsge.calculateLoss`

Written by Kenneth Å!terhagen Paulsen

► **setParametersInUse** ↑

`obj = setParametersInUse(obj,parameters)`

Description:

Indicate that the parameter is in use even if the parser has failed to detect that.

Input:

- obj : An object of class nb_dsge.
- parameters : A cellstr with the parameters to indicate are in use.

Output:

- obj : An object of class nb_dsge.

Written by Kenneth Å!terhagen Paulsen

► **setPrior** ↑

`obj = setPrior(obj,prior)`

Description:

Set priors of a vector of nb_dsg objects. The prior input must either be a struct or nb_struct with same size as obj or have size 1.

The provided struct will be added to the property options as the field prior.

The supported prior distributions for the NB Toolbox are:

- For backing out from mean and std. See the mean and variance only supported distribution listed in the static method nb_distribution.parametrization.
- For the parametrized prior distribution option see the help on the type property of the nb_distribution class, as well as the help on the parameters property for how many parameters each distribution takes.

Caution : If no priors are set maximum likelihood is done when the estimate method is called. Initial values must then be set by the nb_model_generic.assignParameters method.

Input:

- obj : A vector of nb_dsg objects.
- prior : One of;
 - > A struct or a nb_struct with either matching numel of obj or size 1. If size is 1 all models gets the same prior settings.
 - For each element of the struct array:
 - > The fieldnames should be names of the parameters to estimate.
 - > The fields must be one of the following:
 - * Uniform prior:
 {startValue,lowerBound,upperBound}
 - * Prior distribution backed out from mean and standard deviation:
 {startValue,priorMean,priorSTD,'distribution'}
 - * Same as above, but adds a lower bound to the prior distribution:
 {startValue,priorMean,priorSTD,'distribution',lowerBound}
 - * Same as above, but adds a upper bound to the prior distribution:
 {startValue,priorMean,priorSTD,'distribution',lowerBound,...
 upperBound}
 - * Parametrized prior distribution:
 {startValue,parameters,'distribution'}
 - * Same as above, but adds a lower bound to the prior distribution:
 {startValue,parameters,'distribution',lowerBound}
 - * Same as above, but adds a upper bound to the prior distribution:
 {startValue,parameters,'distribution',lowerBound,upperBound}
 where parameters is a 1 x nParam cell array given as the parameters of the selected distribution.
 - > To estimate the the timing of break point use 'break_dateOfBreak' as the fieldname and the field on the the following syntax:
 {lowerBound,upperBound}
 where both lowerBound and upperBound either must be a date as string or a nb_date object. Example:
 prior.break_2001Q1 = {'1999Q1','2001Q1'};

> A nParam x 1 nb_distribution object. Set the name property of the nb_distribution objects to link it to a parameter and use the userData property to set the starting value of the optimization (if it is empty it will use the prior mean as the starting value).

Output:

- obj : A vector of nb_dsg objects with the priors set.

See also:

[nb_model_generic.assignParameters](#), [rise_generic.setup_priors](#)
[nb_model_estimate.set](#)

Written by Kenneth Sæterhagen Paulsen

► **setRisePosteriorSettings** ↑

obj = setRisePosteriorSettings(obj,varargin)

Description:

Set setting on how to sample from the posterior.

Inputs.

- obj : A 1x1 nb_dsg object.

Input:

- 'burnin' : The number of draws to burn at the start of the sampling process.
- 'thin' : Give 1 to keep every draw, 2 to keep every second draw, etc.
- 'algorithm' : Either;
 - > 'mh_sampler' : Uses the mh_sampler function of the RISE toolbox
 - > 'rff_sampler' : Uses the rrf_sampler function of the RISE toolbox

Output:

obj : The object itself with the options reset.

Written by Kenneth Sæterhagen Paulsen

► **setSystemPrior** ↑

```
obj = setSystemPrior(obj,varargin)
```

Description:

Set system priors used for estimation.

The system priors implemented are described in the paper Andrle and Benes (2013), "System Priors: Formulating Priors about DSGE Models'', Properties"

Input:

- obj : A scalar object of class nb_dsge.

Optional input:

- 'irf' : System priors on IRFs. A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1.
 4. Prior function as a function_handle. E.g.
@(x) log(nb_distribution.normal_pdf(x,0,1)), i.e. it must return the log prior density.

See nb_distribution.parametrization or
nb_distribution.qestimation to back out the
hyperparameters of a distribution given some moments.

E.g. {'E_X','X',1,@(x) log(nb_distribution.normal_pdf(x,0,1))}

- 'cov' : System priors on the correlation matrix. A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'X','X',0,@(x) log(nb_distribution.invgamma_pdf(x,1,1))},
i.e. prior on the variance of 'X'.

> 'corr' : Same as for 'cov', but now the system prior is on the correlation matrix. (Be aware that the example above makes no sense in this case as the contemporaneous correlation with itself is always 1!)

> 'fevd' : System priors on forecast error variance decomposition.
A N x 4 cell, where the format is as in the 'irf' case.

Output:

- obj : A scalar object of class nb_dsge, where the systemPrior options has been assign a function_handle that evaluates the wanted priors.

See also:

`nb_dsgc.setPrior`, `nb_model_generic.estimate`, `nb_model_sampling.sample`
`nb_model_sampling.sampleSystemPrior`

Written by Kenneth Åkerhagen Paulsen

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of `nb_model_generic` objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- `obj` : A vector of `nb_model_generic` objects
- `'packages'` : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1','...','shock_group_name_N';  
'shock_name_1' ,..., 'shock_name_N'}
```

Where '`shock_name_x`' must be a string with a shock name or a cellstr array with the shock names. E.g '`E_X_NW`' or `{'E_X_NW','E_Y_NW'}`.

Caution: Two special identifiers can be used; '`Initial Conditions`' and '`Steady-state`'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- `'variables'` : The variables to decompose. Default is the dependent variables defined by the first model.
- `'startDate'` : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- `'endDate'` : End date of the decomposition. As a string. If

different models has different frequency this date will be converted.

- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. [0.3, 0.5, 0.7, 0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.

- `decompBand` : A nested structure of `nb_ts` objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- `plotter` : A $1 \times nModel$ vector of `nb_graph_ts` objects. Use the `graph` method to produce the graphs or use the `nb_graphPagesGUI` on each element of the graph objects.

See also:

[nb_dsge.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **simulate** ↑

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `nSteps` : Number of simulation steps. As a 1×1 double. Default is 100.

Optional inputs:

- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- `'bounds'` : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - `'shock'` : Name of the shock to match the restriction on the bounds of the given variable.
 - `'lower'` : The lower bound of the selected variable. Either a 1×1 double value or a function handle to a probability distribution to draw from.
 - `'upper'` : The upper bound of the selected variable. Either a 1×1 double value

or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not

the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsg models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to

the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.

- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulateStructuralMatrices** ↑

[Alead, A0, Alag, B] = simulateStructuralMatrices(obj, method, headings)

Description:

```
Alead*y(+1) + A0*y + Alag*y(-1) + B*e
```

Input:

- obj : An object of class nb_dsge.
- method : Either 'mean', 'median', 'var', 'std', 'skewness', 'kurtosis' or 'sim' (default). 'sim' will return the simulation where the pages are the different draws.
- headings : Give true to add table headings to the outputs. Not supported for method set to 'sim'. Default is false.

Output:

- [Alead,A0,Alag,B] : The structural representation of the DSGE model. Ax has size nEndo x nEndo x N, and B has size nEndo x nExo x N. N is equal to options.uncertain_draws if method is equal to 'sim', otherwise 1.

See also:

[nb_dsge.declareUncertainParameters](#)

Written by Kenneth Sæterhagen Paulsen

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.
- Optional inputs;
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to

true.

- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.

- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)),

where x is the variable along the first dimension. In the lower triangular part the you can find $\text{cov}(x(-i), y)$, where x is the variable along the first dimension.

E.g: You have two variables x and y , then to find $\text{cov}(x, y(-i))$ you can use $\text{getVariable}(\text{varargin}\{i\}, 'y', 'x')$, while to get $\text{cov}(x(-i), y) (= \text{cov}(x, y(+i)))$ you can use $\text{getVariable}(\text{varargin}\{i\}, 'x', 'y')$. (This example only works if the output is of class nb_cs)

See also:

[nb_dsge.graphCorrelation](#), [nb_dsge.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve** ↑

```
obj = solve(obj)
obj = solve(obj, numAntSteps, shockProperties)
obj = solve(obj, varargin)
```

Description:

Solve estimated/calibrated model(s) represented by nb_dsge object(s).

This method uses the RISE model solver if the nb_dsge model represent a RISE object, and it uses the dynare developed solver if the underlying object is a dynare solved DSGE model. All credits should go to the authors of those codes.

If the model file is of the .nb extension the solver should be credited the author of this function, but again this solver is based on papers published by the dynare team and other. See:

```
> "Solving rational expectations models at first order: what Dynare does"
   by Sebastian Villemot
> "Loose commitment in medium-scale macroeconomic models:
   Theory and an application" by Debortoli, Maih and Nunes (2010).
> "Conditional forecast in DSGE models." by Maih (2010).
```

Input:

- obj : A vector of nb_model_generic objects.

Optional inputs:

Only two extra inputs (and the varargin{1} input is numeric):

- varargin{1} : See nb_dsge.help('numAntSteps').

```

- varargin{2} : See nb_dsge.help('shockProperties') .

OR (optionName, optionValue pairs):

- 'numAntSteps'      : See nb_dsge.help('numAntSteps') .

- 'silent'           : See nb_dsge.help('silent') .

- 'shockProperties' : See nb_dsge.help('shockProperties') .

- 'waitbar'          : Set to nb_waitbar5 object. Only if numel(obj) > 1 .
It will use the maxIterations2 and status2
properties.

```

Can also set all options of the options property. Same as
 nb_model_estimate.set.

Output:

```

- obj    : A vector of nb_model_generic objects, where the
solved model(s) is/are stored in the property
solution.

- solved : If nargout == 2 this method will not throw an error in the
case an element does not solve, instead it will set the
corresponding element of this output to false, and return the
un-solved nb_dsge object array.

```

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_dsge.solveNormal](#)
[nb_dsge.solveExpanded](#)

Written by Kenneth Å!terhagen Paulsen

► **solveBalancedGrowthPath ↑**

obj = solveBalancedGrowthPath(obj)

Description:

Solve for the balanced growth path of the model.

Input:

- obj : An object of class nb_dsge.

Output:

- obj : An object of class nb_dsge.

See also:

[nb_dsge.stationarize](#)

Written by Kenneth Sæterhagen Paulsen

► **solveBreakPoints** ↑

```
[AB,BB,CB,CEB,ssB,JACB,err] = nb_dsge.solveBreakPoints(options,...  
beta,A,B,C,CE,ss,JAC,silent)
```

Description:

Solve for all the break-points regimes.

Input:

- options : See the estOptions property of the nb_dsge class.
- beta : A nParam x 1 double with the parameter values in the main regime.
- A : State transition matrix in main regime.
- B : Constant term
- C : The impact of shock matrix in main regime.
- CE : The impact of anticipated shock matrix in main regime.
- ss : The steady-state in the main regime.
- JAC : The jacobian of the main regime.

Output:

- AB : A 1 x nBreaks + 1 cell with the state transition matrices of all regimes.
- BB : A 1 x nBreaks + 1 cell with the constant term.
- CB : A 1 x nBreaks + 1 cell with the impact of shock matrices of all regimes.
- CEB : A 1 x nBreaks + 1 cell with the impact of anticipated shock matrices of all regimes.
- ssB : A 1 x nBreaks + 1 cell with the steady-state of all regimes.
- JACB : A 1 x nBreaks + 1 cell with the jacobian matrices of all regimes.
- err : Non-empty if an error is thrown. If this output is not returned a standard error is thrown in the command window.

Written by Kenneth Sæterhagen Paulsen

► **solveExpanded** ↑

```
tempSol = nb_dsg.e.solveExpanded(model, results, opt, numAntSteps, ...
                                  shockProperties)
```

Written by Kenneth Sæterhagen Paulsen

► **solveForEpilogue** ↑

```
[A,B,parser] = nb_dsg.e.solveForEpilogue(parser, solution, H, D, Alead, Alag, algorithm)
```

Description:

Solve for the epilogue when it is remove during normal solving.

See also:

[nb_dsg.e.looseOptimalMonetaryPolicySolver](#)
[nb_dsg.e.optimalMonetaryPolicySolver](#), [nb_dsg.e.rationalExpectationSolver](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_dsg.e.solveNormal(results, opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveOneIteration** ↑

```
[A,B,C,CE,ss,paramV,err] = nb_dsg.e.solveOneIteration(options, results, obs)
```

Description:

Solve model given current level of trends.

Input:

- options : See the `estOptions` property of the `nb_dsg.e` class.
- results : See the `results` property of the `nb_dsg.e` class.
- obs : A struct with the current values of the endogenous variables of the model.

Output:

- A : State transition matrix.
- B : The impact of exogenous variables matrix.
- C : The impact of shock matrix.
- CE : The impact of anticipated shock matrix.
- ss : The steady state, or point of approximation.
- err : Non-empty if an error is thrown. If this output is not return a standard error is thrown in the command window.

Written by Kenneth Sæterhagen Paulsen

► solveOneRegime ↑

```
[A,B,C,CE,ss,JAC,err] = nb_dsge.solveOneRegime(options,beta,skipSolve,silent)
```

Description:

Solve one regime.

Input:

- options : See the estOptions property of the nb_dsge class.
- beta : The parameters of the current regime.
- skipSolve : Skip solving for A, B and BE. (They are returned as [])

Output:

- A : State transition matrix.
- B : Constant term.
- C : The impact of shock matrix.
- CE : The impact of anticipated shock matrix.
- ss : Steady state.
- JAC : Jacobian.
- err : Non-empty if an error is thrown. If this output is not return a standard error is thrown in the command window.

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_dsgc.solveRecursive(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveSteadyState** ↑

```
[obj,err] = solveSteadyState(obj,varargin)
```

Description:

If the 'steady_state_solve' is set to true, the solution will be solved for analytically. If the 'steady_state_file' option is not empty, it is assumed to provided the solution (see comment below also!). If neither of them is used the steady-state is assumed to be 0 for all variables.

Caution: This function is automatically called inside
nb_dsgc.checkSteadyState if the option 'steady_state_solve' is
set to true and is not called by the user manually!

Caution: If the 'steady_state_file' option is not empty and the option
'steady_state_solve' is set to true, the provided file
will be used as the starting point for the solver, i.e.
use this file can be used to set initial values for the
solution of the steady-state.

Caution: If some variable is not been given any value in the
steady-state file it is assumed that it is 0 in steady-state.

Input:

- obj : An object of class nb_dsgc.

Optional inputs:

- 'silent' : See nb_dsgc.help('silent')
- 'steady_state_file' : See nb_dsgc.help('steady_state_file')
- 'steady_state_solve' : See nb_dsgc.help('steady_state_solve')
- 'solver' : See nb_dsgc.help('solver')

Output:

- obj : An object of class nb_dsgc, where the solution of the
steady-state can be found at obj.solution.ss.
- err : If two outputs are called for a potential error message is not
thrown, but instead return as a string. Is empty if not error
occure. Caution: This is only the case if numel(obj) == 1.

See also:

[nb_dsgc.help](#), [nb_dsgc.checkSteadyState](#), [nb_dsgc.solveSteadyStateStatic](#)

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

obj = solveVector(obj)

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **split** ↑

objVec = split(obj)

Description:

Split a NB toolbox or RISE solved DSGE model into separates objects for each regime.

Caution : In the case of a RISE object, calling solve again on the object will result in the a object with more regimes!

Input:

- obj : An object of class nb_dsgc, either solved with the NBToolbox or RISE.

Output:

- objVec : A vector of nb_dsgc object representing each regime.

See also:

nb_dsg.e.solve

Written by Kenneth Sæterhagen Paulsen

► stateSpace ↑

```
sol = nb_dsg.e.stateSpace(par,estStruct)
```

Description:

Returns a state-space representation of a DSGE model.

Observation equation:

$$y(t) = H \cdot x(t)$$

State equation:

$$x(t) = A \cdot x(t-1) + B \cdot z(t) + C \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I .

Input:

- par : Current vector of the estimated parameters.
- estStruct : See the nb_dsg.e.getObjectiveForEstimation method.

Output:

- sol : A struct with the fields:
 - > H : Observation matrix.
 - > A : State transition matrix.
 - > B : Constant term.
 - > C : B is impact of shock matrix.
 - > x : The "a priori" state estimate (after the new measurement information is included).
 - > P : Covariance of the state vector "a priori" estimate.
 - > Pinf : Covariance of the state vector "a priori" estimate used during diffuse steps.
 - > options : Same as the input options + the ss fields will be added.
 - > err : A char with the error message. Empty if no error have occurred.
 - > ss : The steady-state of the model.

See also:

[nb_kalmanLikelihoodMissingDSGE](#), [nb_kalmanlikelihoodDSGE](#)
[nb_model_generic.estimate](#), [nb_dsge.objective](#)

Written by Kenneth Å!terhagen Paulsen

► **stateSpaceBreakPoint** ↑

```
sol = nb_dsge.stateSpaceBreakPoint(par,estStruct)
```

Description:

Returns a state-space representation of a break-point DSGE model.

Observation equation:

$$y = H \cdot x(t)$$

State equation:

$$x(t) = ss(t) + A(t) \cdot (x(t-1) - ss(t)) + B(t) \cdot z(t) + C(t) \cdot u(t)$$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I .

Input:

- par : Current vector of the estimated parameters.
- estStruct : See the `nb_dsge.getObjectiveForEstimation` method.

Output:

- sol : A struct with the fields:
 - > H : Observation matrix.
 - > A : State transition matrix.
 - > B : Constant term.
 - > C : B is impact of shock matrix.
 - > x : The "a priori" state estimate (after the new measurement information is included).
 - > P : Covariance of the state vector "a priori" estimate.
 - > Pinf : Covariance of the state vector "a priori" estimate used during diffuse steps.
 - > options : Same as the input options + the states and ss fields will be added.
 - > err : A char with the error message. Empty if no error have occurred.

```
> ss      : The steady-state of the model.  
> states  : The states used for each step of the Kalman filter.
```

See also:

[nb_kalmanlikelihoodDSGE](#), [nb_model_generic.estimate](#), [nb_dsge.objective](#)

Written by Kenneth Åkerhagen Paulsen

► **stateSpaceTVP ↑**

```
sol = nb_dsge.stateSpaceTVP(par,estStruct)
```

Description:

Returns a state-space representation of a DSGE model.

Observation equation:
 $y(t) = H*x(t)$

State equation:
 $x(t) = A(t)*x(t-1) + B(t)*z(t) + C(t)*u(t)$

Where $u \sim N(0, I)$ meaning u is gaussian noise with covariance I .

Input:

- par : Current vector of the estimated parameters.
- estStruct : See the [nb_dsge.getObjectiveForEstimation](#) method.

Output:

- sol : A struct with the fields:
 - > H : Observation matrix.
 - > A : State transition matrix.
 - > B : B is impact of shock matrix.
 - > x : The "a priori" state estimate (after the new measurement information is included).
 - > P : Covariance of the state vector "a priori" estimate.
 - > Pinf : Covariance of the state vector "a priori" estimate used during diffuse steps.
 - > options : Same as the input options + the ss fields will be added.

```
> err      : A char with the error message. Empty if no error have  
occured.  
  
> ss       : The steady-state of the model.
```

See also:

[nb_kalmanlikelihoodDSGE](#), [nb_model_generic.estimate](#), [nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

► **stationarize ↑**

```
obj = stationarize(obj,varargin)
```

Description:

Stationarize model.

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'waitbar' : Give this to trigger waitbar during stationarization.

Output:

- obj : An object of class nb_dsge.

See also:

Written by Kenneth Sæterhagen Paulsen

► **struct ↑**

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_dsgen.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **structuralMatrices2JacobianNB ↑**

```
jacobian = nb_dsgen.structuralMatrices2JacobianNB(Alead,A0,Alag,B,...  
leadCurrentLag)
```

Description:

Derive the jacobian given the structural matrices.

On the form: $A_{lead} \cdot y(t+1) + A_0 \cdot y + A_{lag} \cdot y(t-1) + B \cdot \epsilon = 0$

Input:

- Alead : The structural matrix of the endogenous variables in the period t+1.
- A0 : The structural matrix of the endogenous variables in the period t.
- Alag : The structural matrix of the endogenous variables in the period t-1.
- B : The structural matrix of the exogenous variables in period t.
- leadCurrentLag : A nEndo x 3 double with the lead, current and lag indices of the endogenous variables of the model.

Output:

- jacobian : A matrix with the jacobian of the DSGE model.

Written by Kenneth Sæterhagen Paulsen

► **systemPriorObjective ↑**

```
fval = nb_dsgen.systemPriorObjective(par,estStruct)
```

Description:

The objective to sample from to get the final prior distribution which also takes into account the system priors.

Input:

- par : Current vector of the estimated parameters.
- estStruct : A struct with at least the following fields;
 - > indPar : Index for where in the full parameter vector to find the estimated parameters.
 - > beta : A vector of the values of all parameters. As a nParam x 1.
 - > model : The obj.parser property of the nb_dsg class.
 - > options : The obj.estOptions property of the nb_dsg class.
(After calling getEstimationOptions, i.e. calling the estimate method.)
 - > y : A nObservables x nPeriods double with the data to estimate the model on. The data may contain missing observations.
 - > z : Not in use. To make it generic to other classes of models.
 - > observables : A logical vector. An element is true if the variable is observable.

Output:

- fval : Value of minus the log system prior at the given parameters.

See also:

[nb_model_sampling.sampleSystemPrior](#)

Written by Kenneth Å!terhagen Paulsen

► template ↑

```
options = nb_dsg.template()  
options = nb_dsg.template(num)
```

Description:

Construct a struct which can be provided to the nb_dsg class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► testForecastRestrictions ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillty intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is

'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargout{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargout{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_dsgc.graphCorrelation](#), [nb_dsgc.parameterDraws](#)

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **tracePlot ↑**

```
[DB,plotter] = tracePlot(obj)
```

Description:

Redirect to checkPosteriors.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_SAMPLING

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and

the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_dsg.e.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

obj = nb_model_estimate.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_dsg.e.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update** ↑

```
obj          = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve
> 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **updateParams** ↑

```
model = nb_dsge.updateParams(model, results)
```

Description:

Update parameter values of a RS-DSGE model.

Written by Kenneth Sæterhagen Paulsen

► **updateSolution** ↑

```
[A,B,C,ss,p,err] = nb_dsge.updateSolution(options,results,xu,p,tt)
```

Description:

Resolve the model when dealing with stochastic trend.

See also:

[nb_kalmanSmotherUnivariateStochasticTrendDSGE](#)
[nb_kalmanLikelihoodUnivariateStochasticTrendDSGE](#)
[nb_kalmanSmotherDiffuseStochasticTrendDSGE](#)
[nb_kalmanLikelihoodDiffuseStochasticTrendDSGE](#)

Written by Kenneth Sæterhagen Paulsen

► **variance_decomposition** ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the

method nb_var.set_identification for more.

- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidenc/probability bands. Default is '''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_dsge.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **writeModel2File ↑**

```
writeModel2File(obj,filename,varargin)
```

Description:

Write model to .nb file.

Input:

- obj : A scalar nb_dsge object.
- filename : A one line char with the file name to write to. Must have .nb or no extension.

Optional input:

```

- 'parameters' : One of:
    > 'file'   : Write the parameters to a seperate file
                  with name [filename '_param.mat'].
                  Default.
    > 'block'  : Write the assignment of the parameters
                  to the parameterization block of the
                  model file.
    > 'off'    : Don't write the parameter values to any
                  format.

- 'precision' : See nb_num2str. Default is 14.

- 'steady_state' : One of:
    > 'file'   : Write the intitial condition for solving
                  the steady state to a seperate file
                  with name [filename '_ss.mat'].
                  Default.
    > 'block'  : Write the assignment of the intitial
                  condition for solving the steady state
                  to the steady_state_model block of the
                  model file.
    > 'off'    : Don't write the intitial condition for
                  solving the steady state to any format.

```

Output:

Examples:

See also:

Written by Kenneth Sæterhagen Paulsen

► **writePDF** ↑

```

writePDF(obj,filename)
writePDF(obj,filename,type)

```

Description:

Write the equations of the model to PDF.

Input:

```

- obj      : An object of class nb_dsgen.

- filename : The filename of the saved pdf file. As a one line char.
             Extension is ignored.

- type     : Either 'names' or 'values'. Default is 'names'. If 'values'
             is given the parameters will be substituted with their
             values instead of their name.

```

Examples:

```
obj.writePDF('test')
```

See also:

[nb_dsge.writeTex](#), [nb_model_generic.assignTexNames](#)

Written by Kenneth S. Paulsen

► **writeTex ↑**

```
code = writeTex(obj)
code = writeTex(obj,filename,type)
writeTex(obj,filename,type)
```

Description:

Write latex code of model equations.

Input:

- obj : An object of class nb_dsge.
- filename : Give the name of the written .tex file. If not given or empty no file will be written. Extension is ignored.
- type : Either 'names' or 'values'. Default is 'names'. If 'values' is given the parameters will be substituted with their values instead of their name.

Output:

- code : A cellstr with the latex code of the model equations.

See also:

[nb_dsge.writePDF](#), [nb_model_generic.assignTexNames](#)

Written by Kenneth Sæterhagen Paulsen

■ **nb_ecm**

Go to: [Properties](#) | [Methods](#)

A class for estimation of error correction models.

Superclasses:

[nb_model_generic](#)

Constructor:

```
obj = nb_ecm(varargin)

Optional input:
```

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_ecm object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_ecm.help](#), [nb_ecm.template](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- addAutoName
- addAutoNameIfLocal
- addID
- addIDIfLocal
- dependent
- endogenous
- estOptions
- exogenous
- forecastOutput
- name
- options
- parameters
- reporting
- residuals
- results
- solution
- transformations
- userData

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj =

```
set(obj,'dependent',{'Var1','Var2'}); or use the  
<className>.template() method.
```

Caution: For nb_dsgc models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t$, $e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t$, $u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- [getScore](#)
- [getVariablesList](#)
- [grouped_decomposition](#)
- [handleMissing](#)
- [initialize](#)
- [irf](#)
- [isFiltered](#)
- [isMixedFrequency](#)
- [isStateSpaceModel](#)
- [isbayesian](#)
- [isforecasted](#)
- [jointPredictionBands](#)
- [mcfRestriction](#)
- [monteCarloFiltering](#)
- [parameterIntervals](#)
- [plotForecastDensity](#)
- [plotMCFDistTest](#)
- [plotPosterioriors](#)
- [print](#)
- [print_estimation_results](#)
- [set](#)
- [simulate](#)
- [simulatedMoments](#)
- [solveLevel](#)
- [solveRecursive](#)
- [struct](#)
- [testForecastRestrictions](#)
- [theoreticalMoments](#)
- [uncondForecast](#)
- [update](#)
- [getSolution](#)
- [graphCorrelation](#)
- [handleCondInfo](#)
- [help](#)
- [interpretForecast](#)
- [isDensityForecast](#)
- [isMS](#)
- [isNB](#)
- [isStatic](#)
- [isestimated](#)
- [issolved](#)
- [loadPosterior](#)
- [mincerZarnowitzTest](#)
- [parameterDraws](#)
- [plotForecast](#)
- [plotMCF](#)
- [plotMCFValues](#)
- [plotPriors](#)
- [printCov](#)
- [removeObservations](#)
- [shock_decomposition](#)
- [simulateFromDensity](#)
- [solve](#)
- [solveNormal](#)
- [solveVector](#)
- [template](#)
- [testParameters](#)
- [uncertaintyDecomposition](#)
- [unstruct](#)
- [variance_decomposition](#)

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_ecm.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_ecm.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_ecm.forecast](#), [nb_ecm.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_ecm.forecast`, `nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast`, `nb_ecm.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_ecm.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast ↑**

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'      : The length of the bins og the domain of the density.
  Either;
    > []       : The domain will be found. See
      nb_getDensityPoints (default is that 1000
      observations of the density is stored).
    > integer   : The min and max is found but the length
      of the bins is given by the provided
      integer.
    > cell       : Must be on the format:
      {'Var1',lowerLimit1,upperLimit1,binsL1;
       'Var2',lowerLimit2,upperLimit2,binsL2;
       ...}
    - lowerLimit1 : An integer, can be nan,
      i.e. will be found.
    - upperLimit1 : An integer, can be nan,
      i.e. will be found.
    - binsL1      : An integer with the
      length of the bins. Can
      be nan. I.e. bins length
      will be adjusted to
      create a domain of 1000
      elements.

Caution : The variables not included will be given
          the default domain. No warning will be
          given if a variable is provided in the
          cell but not forecast by the model. (This
          because different models can forecast
          different variables)

Caution : The combineForecast method of the
          nb_model_group class is much faster if the
          lower limit, upper limit! Or else
          it must simulate new draws and do kernel
          density estimation for each model again
          with the shared domain of all models
          densities.

- 'startDate'   : The start date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
        the start date of the recursive forecast.
        Default is to start from the first possible
        date.

- 'endDate'     : The end date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
the ergodic steady-state for
MS-model, and in the last regime
for break-point models.

- 'startingProb' : Either a double or a char:
 - > 'zero' : Start from zero on all variables
(Except for the deterministic
exogenous variables)
 - > [] : If the model is filtered the
starting value is calculated
on filtered transition
probabilities. Otherwise the
ergodic mean is used.
 - > double : A nPeriods x nRegime or a 1 x
nRegime double. nPeriods refer to
the number of recursive periods to
forecast.
 - > 'ergodic' : Start from the ergodic transition
probabilities.
- 'stabilityTest' : Give true if you want to discard the parameter draws
that give rise to an unstable model. Default
is false.
- 'states' : Either an integer with the regime to
forecast/simulate in, or an object of class nb_ts
with the regimes to condition on. The name of the
variable must be 'states'.

Caution: For break-point models this is decided by
the dates of the breaks, and cannot be set
by this option!

- 'compareToRev' : Which revision to compare to. Default is final
revision, i.e. []. Give 5 to get fifth revision.
must be an integer. Keep in mind that this number
should be larger than the nSteps input.
- 'compareTo' : Sometime you want to evaluate the forecast of one
variable against another variable which is not
part of the model. E.g. if you use the first release
of a variable and want to compare it against the
final release. To do this you can give a cellstr
with size n x 2, where n is the number of variables
to change the the actual observations to test
against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities' : true or false. true if you want to do a
kernel density estimation of the density
forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting
them by a fitted model. Only when the 'condDB'
input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_ecm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_ecm.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_ecm.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_ecm.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a $N \times 3$ cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs \times nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_ecm.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_ecm.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables** ↑

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as

fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

`nb_model_generic.forecast`, `nb_model_generic.constructScore`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getResidual** ↑

`residual = getResidual(obj)`

Description:

Get residual from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- `residual` : Either a `nb_ts` or a `nb_cs` object with `residual(s)` stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualGraph** ↑

`plotter = getResidualGraph(obj)`

Description:

Get residual graph from the given estimator. The returned will be an object of class `nb_graph`, which you can call the `graphInfoStruct` method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- `obj` : An object of class `nb_model_generic` (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph`. Use the `graphInfoStruct` method or the `nb_graphInfoStructGUI` class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of `nb_model_generic` object

Input:

- `obj` : A vector of `nb_model_generic` objects

Output:

- `residualNames` : A cellstr with the unique residual names of all the models

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method `solve`.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
- 'RMSE' : One over root mean squared error
- 'MSE' : One over mean squared error
- 'MAE' : One over mean absolute error
- 'MEAN' : One over mean error
- 'STD' : One over standard error of the forecast error
- 'ESLS' : Exponential of the sum of the log scores
- 'EELS' : Exponential of the mean of the log scores
- 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.

- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_ecm.simulatedMoments](#), [nb_ecm.theoreticalMoments](#)
[nb_ecm.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **grouped_decomposition** ↑

[dec, plotter] = grouped_decomposition(obj, varargin)

Description:

Produce historical decomposition of a ECM model where contribution related to one variable is grouped automatically.

Input:

- obj : An object of class nb_ecm.

Optional input:

- 'method' : One of
 - > 'shock_decomposition' : This uses the shock_decomposition method to do the decomposition, and then it aggregates all the contributions that relates to one variable, e.g. aggregates the contributions from different lags and the lagged level of the endogenous variables. Default method.

> 'recursive_forecast' : This method uses recursive forecasting to produce the decomposition. It creates a baseline recursive forecast. In this baseline it condition on all right hand side variables, where all level variable are kept at 0 and all difference variables are kept at their mean. Then it sets one variable at the time to its actual value, and produce a recursive forecast in this case. The contribution from this variable are then the difference between this forecast compared to the baseline (both being transformed using the function specified by the 'transformation' input).

This method are wrong for the first periods in the 'diff' transformation is used, but it converges to the correct solution at the end. If 'level' is chosen it does not converge to the true solution!

- 'transformation' : Either 'level' (no transformation) or 'diff' (difference).

Caution: If the dependent variable is in log 'diff' will give the growth rate.

- 'nLags' : The number of lags used by the diff operator. Default is 1.

- 'inputVars' : true or false. Set it to true if you want to further decompose the ECM model into contribution of each input variable instead of each model variable. Use the nb_modelData.createVariables to be able to do this. Default is false. This will automatically switch to the method 'recursive_forecast'!

Caution: The the expressions are non-linear this will only be an "linear" approaximation of the contribution of each series. The approximation error will end up the 'Rest' variable.

- 'seasonalPattern' : Set it to true if you want to keep the seasonal pattern in the level data when constructing the "steady state" values of the level variables. Used only when 'method' is set to 'recursive_forecast'. false is default.

- 'modelCond' : A nb_ecm object which has produced conditional forecast.

Output:

- dec : The final decomposition as a nb_ts object with size nObs x (nEndo + nExo + 1). 'Rest' includes the residual.

- plotter : A nb_graph_ts object that can produce a graph of the

decomposition. Use the graph method or the nb_graphPagesGUI class.

Examples:

```
\NBTOOLBOX\Econometrics\test\test_nb_ecm.m  
\NBTOOLBOX\Econometrics\test\test_nb_ecm_dec.m
```

See also:

[nb_model_generic.shock_decomposition](#)

Written by Kenneth Sæterhagen Paulsen

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

```
- ret : true or false. true if the estimation settings is so that the  
model handle missing data.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_ecm.help  
helpText = nb_ecm.help(option)  
helpText = nb_ecm.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

```
- option : Either 'all' or the name of the option you want to get the  
help on.  
  
- maxChars : The max number of chars in the printout of (approx). This  
applies to the second column of the printout. Default is  
40.
```

Output:

```
- helpText : A char with the help.
```

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize ↑**

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

```
- objClass : The name of any of the subclasses of the nb_model_generic  
class. As a string.  
  
- template : The output from the template method of each subclass of the  
nb_model_generic class. Multiple model objects will be  
initialized if the num input that method is greater than one.
```

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you

need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.

- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_ecm.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsg objects.
- 'compare' : Give true if you have a scalar nb_dsg model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.
- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.
- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.

I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

```

- 'fanPerc'          : This options sets the error band percentiles of the
                      graph, when the 'perc' input is empty. As a 1 x
                      numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
                      Default is 0.68.

- 'foundReplic'      : A struct with size 1 x replic. Each element
                      must be on the same format as obj.solution. I.e.
                      each element must be the solution of the model
                      given a set of drawn parameters. See the
                      parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare'       : A struct on the same format as the irfs output from
                      this function with the IRFs to compare to.

- 'levelMethod'       : One of the following:

> 'cumulative product'           : cumprod(1 + X,1)
> 'cumulative sum'              : cumsum(X,1)
> 'cumulative product (log)'    : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)'       : cumprod(1 + X/100,1)
> 'cumulative sum (%)'          : cumsum(X/100,1)
> 'cumulative product (log) (%)': log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)': exp(cumsum(X/100,1))
> '4 period growth (log approx)': nb_msum(X,3)

- 'method'             : The selected method to create error bands.
                      Default is ''. See help on the method input of the
                      nb_model_generic.parameterDraws method for more
                      this input. An extra option is:

> 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent
                  variables. See nb_var.set_identification. Of
                  course this option only makes sense for VARs
                  identified with sign restrictions.

- 'normalize'          : A 1 x 3 cell. First element must be the variable
                      name as a string. The second element must be the
                      value to normalize to, as an integer. While the
                      third element is the period to be normalized, if
                      set to inf, it will normalize the max impact
                      period.

E.g. {'Var',1,2}, {'Var',1,inf}

Caution: 2 means observation 2 of the IRF, and as
          the IRF start at period 0, this means that

```

observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.

Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- 'parallel' : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.

```

- 'shocks'           : Which shocks to create impulse responses of.
                      Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign'             : Sign of the impulse. Either 1 or -1. Can also be
                      a vector of the same size as 'shocks' input.

- 'stabilityTest'   : Give true if you want to discard the draws
                      that give rise to an unstable model. Default
                      is false.

- 'startingProb'    : Either a double or a char (Only for Markov-switching
                      models):

    > scalar          : Select the the regime to take the
                          initial transition probabilities
                          from.

    > double           : A draws x nRegime or a 1 x
                          nRegime double. draws refer to
                          the number set by the 'draws'
                          input.

    > 'ergodic'        : Start from the ergodic transition
                          probabilities. Default for 'irf'.

    > 'random'         : Randomize the starting values
                          using the simulated starting
                          points. Default for 'girf'.

- 'startingValues'  : Either a double or a char:

    > double           : A nVar x 1 double.

    > 'steadystate'    : Start from the steady state. If you are dealing
                          with a MS-model or a break-point model you can
                          indicate the regime by 'steadystate(regime)'. E.g.
                          to start in regime 1 give; 'steadystate(1)'. For
                          MS - models the default is the ergodic mean
                          (default as long as 'girf' is not selected as
                          'type'), while for break-point models it is to
                          start from the first regime.

    > 'zero'            : Start from zero on all variables.

    > 'random'          : Randomize the starting values using the simulated
                          starting points. Default when 'type' set to
                          'girf'.

- 'states'           : Either an integer with the states to produce irf in,

```

or a double with the states to condition on. Must have size periods x 1 in the last case. Applies to both MS - models and break-point models.

- 'type' : 'girf' or 'irf'. (If empty 'irf' will be used)
 - > 'girf' : Generalized impulse response function calculated as the mean difference between shocking the model with a one period shock (1 std) and no shock at all. Use the 'draws' input to set the number of simulations to use to base the calculation on. This type of irfs must be used for non-linear models.
 - > 'irf' : Standard irf for linear models. Calculated by shocking the model with a one period shock (1 std).
- 'variables' : The variables to create impulse responses of. Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!
- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.'(E_X_NW)'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.
- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!
- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

```
> 'compareShocks' set to true
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

- obj      : If the process has been paused, some temporary output will
              be stored in the object, and if you want to continue at a
              later stage you can take up from this point using this
              returned output. If not paused this output is empty.
```

See also:

[nb_ecm.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

```
- obj : A nb_model_forecast object (matrix)
```

Output:

```
- ret : A logical with same size as obj.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsgen object is filtered or not.

Input:

```
- obj      : An object of class nb_dsgen.
```

```
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is
          'smoothed'.
```

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isMS ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isMixedFrequency ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsg objects!

For nb_dsg objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.
- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_ecm.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x) && f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

```

> 'irf'      : A N x 4 cell, where the elements of each
               row are:
               1. Name of the shock. 'E_X'
               2. Name of the variable. 'X'
               3. Horizon. E.g. 1 or 1:3.
               4. Restriction. E.g. @(x)gt(x,0),
                  @(x)gt(x,0)||lt(x,1), i.e. a
                  function_handle that takes a scalar
                  double as input and a scalar logical as
                  output.

> 'rirf'     : A N x 7 cell, where the elements of each
               row are:
               1. Name of the shock 1. 'E_X'
               2. Name of the variable 1. 'X'
               3. Horizon of variable 1. E.g. 1.
               4. Name of the shock 2. 'E_Y'
               5. Name of the variable 2. 'Y'
               6. Horizon of variable 2. E.g. 2.
               7. Restriction. E.g. @(x,y)gt(x,y),
                  @(x,y)gt(x-y,0)||lt(x-y,1), i.e. a
                  function_handle that takes a two scalar
                  double as inputs and a scalar logical
                  as output.

> 'corr'     : A N x 4 cell, where the elements of each
               row are:
               1. Name of the first variable.
               2. Name of the second variable.
               3. Number of periods to lag the 2. variable.
               4. Same as 4 for 'irf'.

               E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to
               test a restriction on the autocorrelation
               at lag for variable 'Var1'.

               E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to
               test a restriction on the contemporaneous
               correlation between 'Var1' and 'Var2'.

> 'cov'       : Same as for 'corr'.

               E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to
               test a restriction on the contemporaneous
               variance.

> 'ss'        : A N x 2 cell, where the elements of each
               row are:
               1. The expression using any of the
                  endogenous variables of the model.
               2. Same as 4 for 'irf'.

```

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_ecm.monteCarloFiltering](#), [nb_ecm.irf](#)
[nb_ecm.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_ecm.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering ↑**

```
paramD           = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if

the model where solved and the test function returned a value.

- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_ecm.mcfRestriction](#), [nb_ecm.solve](#)
[nb_ecm.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.

```

> 'blockBootstrap'      : Create artificial data by
                           non-overlapping block bootstrap,
                           and then "draw" parameters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'mblockBootstrap'    : Create artificial data by
                           overlapping block bootstrap,
                           and then "draw" parameters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'rblockBootstrap'    : Create artificial data by
                           overlapping random block length
                           bootstrap, and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
                           overlapping random block length
                           bootstrap., and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'copulaBootstrap'    : Uses a copula approach to draw residual
                           that are autocorrelated, Does not handle
                           heteroscedasticity. Only an option for
                           models estimated with classical methods.

> 'posterior'          : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores
 available.
- 'newDraws' : When out of parameter draws, this factor decides how
 many new draws are going to be made. Default is 0.1.
 I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given

as a number less than 1, it will by default be set to 100.

- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:

> 'param' : Default. A struct with the following fields:

 beta : A nPar x nEq x draws double with the estimated parameters.

 sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

 lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

 R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

 factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterIntervals** ↑

ci = parameterIntervals(obj,alpha)

Description:

```
Get confidence intervals (classic) or probability intervals (bayesian).
```

```
Confidence intervals are constructed as:
```

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

```
Probability intervals are constructed as the per
```

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

```
Written by Kenneth Sæterhagen Paulsen
```

```
Inherited from superclass NB_MODEL_GENERIC
```

► **plotForecast** ↑

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

```
Plot forecast.
```

```
Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects
```

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.

```
- increment : Number of periods between the hairs when doing a  
hairy-plot. Default is 1.
```

Optional input:

```
- 'type' : If the history is of some variables are filtered, you can  
use this input to choose to plot the smoothed or updated  
estimates of the filtered variables. Either 'smoothed' or  
'updated'. Default is 'smoothed'.
```

Output:

```
- plotter : An object of class nb_graph. Use the graphSubPlots  
method or the nb_graphSubPlotGUI class.  
  
- plotFunction2Use : A string with the name of the method to call on the  
plotter object to produce the graphs.
```

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will
be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the
provided date. See the dataNames property of the DB
property of the returned plotter object. If the model has
produced nowcast the names will 'horizon0' (a variable lag one
period), 'horizon-1' (a variable lag two periods), etc.

Input:

```
- obj : A 1x1 nb_model_forecast object.  
  
- date : A string or nb_date object with the date of the wanted  
forecast.  
  
- variable : A string with the variable of interest.
```

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the
figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF** ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound, ...
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_ecm.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_ecm.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_ecm.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_ecm.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPriors](#) ↑

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_ecm.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

```
- obj      : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA
```

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

```
- obj : A vector of nb_model_estimate objects.
```

Optional input:

If number of inputs equals 1:

```
- varargin{1} : A structure of fields to be set. See the
               template method of each model class for more.
```

Else:

```
- varargin     : ...,'inputName', inputValue,... arguments.
```

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

```
- obj : A vector of nb_model_estimate objects.
```

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition** ↑

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';  
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the

nb_model_generic.parameterDraws method for more this input. An extra option is:

```
> 'identDraws' : Uses the draws of the matrix with  
      the map from structural shocks to dependent variables.  
      See nb_var.set_identification. Of course this option  
      only makes sense for VARs identified with sign  
      restrictions.  
  
- 'replic'          : The number of simulation for posterior, bootstrap and  
      MC methods. Default is 1. Only used when 'perc' is  
      provided.  
  
      Caution: Will not have anything to say for the method  
      'identDraws'. See the option 'draws' of the  
      method nb_var.set_identification for more.  
  
- 'stabilityTest' : Give true if you want to discard the draws  
      that give rise to an unstable model. Default  
      is false.  
  
- 'parallel'        : Give true if you want to do the shock decomp in  
      parallel. Default is false.  
  
- 'fcstDB'          : An object of class nb_ts with the forecast to  
      decompose. must contain all model variables and  
      shocks/residuals, and must start the period after  
      the estimation/filtering end date or at the  
      estimation/filtering start date. See  
      the nb_model_generic.getForecast method.  
  
- 'type'            : Choose 'updated' or 'smoothed'. 'smoothed' is  
      default.  
  
- 'anticipationStartDate' : Start date of anticipating shocks. As a  
      string. If different models has different  
      frequency this date will be converted.  
  
- 'model2'          : A struct with the solution of the second  
      model to use for decomposition.  
  
- 'secondModelStartDate' : Start date of second model. As a string.  
      If different models has different frequency  
      this date will be converted.
```

Output:

- decomp : A structure of nb_ts objects with the shock
 decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty
 bounds of the shock decomposition for each model at each
 percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the
 graph method to produce the graphs or use the
 nb_graphPagesGUI on each element of the graph objects.

See also:

`nb_ecm.parameterDraws, nb_shockDecomp`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **simulate** ↑

`out = simulate(obj,nSteps,varargin)`

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `nSteps` : Number of simulation steps. As a `1x1 double`. Default is 100.

Optional inputs:

- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- `'bounds'` : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - `'shock'` : Name of the shock to match the restriction on the bounds of the given variable.
 - `'lower'` : The lower bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.
 - `'upper'` : The upper bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps double` matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a `1x1 double` or a `nSteps x 1 double`.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty

the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadyState' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsg models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consists of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the

'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.
 Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
 Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
 Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i},'y','x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i},'x','y'). (This example only works if the output is of class nb_cs)

See also:

[nb_ecm.graphCorrelation](#), [nb_ecm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_ecm object(s).

Input:

- obj : A vector of nb_ecm objects.

Output:

- obj : A vector of nb_ecm objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveLevel** ↑

```
sol = solveLevel(obj)
```

Description:

Convert the standard solution into a solution in only levels.

Input:

- obj : An object of class nb_ecm.

Output:

- sol : A struct on the same format as the solution property, but now the solution is in level variable only (dependent only!).

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_ecm.solveNormal(results,opt,iter)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_ecm.solveRecursive(results,opt,ident)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_ecm.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

options = nb_ecm.template()
options = nb_ecm.template(num)

Description:

Construct a struct which can be provided to the nb_ecm class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillty intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_ecm.graphCorrelation](#), [nb_ecm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';  
 'shock_name_1' , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

obj = uncondForecast(obj,nSteps,varargin)

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_ecm.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_ecm.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update** ↑

```
obj          = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic

- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve
> 'forecast' : Update data, estimate, solve and forecast.

- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.

- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
 - 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
 - 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
- Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the

method nb_var.set_identification for more.

- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidenc/probability bands. Default is '''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N' }
```

Where '`shock_name_x`' must be a string with a shock name or a cellstr array with the shock names. E.g '`E_X_NW`' or `{'E_X_NW','E_Y_NW'}`.

If empty no packing will be done.

Output:

- `decomp` : A structure of `nb_ts` objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the '`output`' option)
- `decompBands` : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as `decomp`.
- `plotter` : A $1 \times nModel$ vector of `nb_graph_cs` objects. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
- `plotterBands`: A $1 \times nModel$ struct. Each field is a $1 \times nVars$ vector of `nb_graph_cs` objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_ecm.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

■ nb_exprModel

Go to: [Properties](#) | [Methods](#)

A class for estimation a model where each variable is allowed to be an expression of different variables, e.g.

```
diff(y(t)) = beta(1)*diff(x(t)) + beta(2)*(log(y(t-1)) - log(x(t-1)))
```

Superclasses:

`nb_model_generic`

Constructor:

```
obj = nb_exprModel(varargin)
```

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_exprModel object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [dependent](#)
- [endogenous](#)
- [estOptions](#)
- [exogenous](#)
- [forecastOutput](#)
- [name](#)
- [options](#)
- [parameters](#)
- [reporting](#)
- [residuals](#)
- [results](#)
- [solution](#)
- [transformations](#)
- [userData](#)

- **[addAutoName](#) ↑**

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#) ↑**

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#) ↑**

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#) ↑**

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#) ↑**

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgen models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.

- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- `getScore`
- `getVariablesList`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`
- `removeObservations`
- `shock_decomposition`
- `simulateFromDensity`
- `solutionFunc`
- `solveNormal`
- `solveVector`
- `template`
- `testParameters`
- `uncertaintyDecomposition`
- `unstruct`
- `variance_decomposition`
- `getSolution`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `irf`
- `isFiltered`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `set`
- `simulate`
- `simulatedMoments`
- `solve`
- `solveRecursive`
- `struct`
- `testForecastRestrictions`
- `theoreticalMoments`
- `uncondForecast`
- `update`

► **`appendData`** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames** ↑

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_exprmodel.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convert`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convert`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

`obj = convert(obj,freq,methods,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_exprModel.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_exprmodel.forecast](#), [nb_exprModel.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_exprmodel.forecast, nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast, nb_exprModel.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

```

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables.
            'dependent' will return the moment of the dependent
            variables. 'dependent' is default.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or
               not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to
             true.

- 'type' : Either 'covariance' or 'correlation'. Default is
            'correlation'.

- 'startDate' : The start date of the calculations. Default is the start
                date of the options.data property. Only for time-series!
                Must be a string or a nb_date object.

- 'endDate' : The end date of the calculations. Default is the end
               date of the options.data property. Only for time-series!
               Must be a string or a nb_date object.

- 'demean' : true (demean data during estimation of the
              autocovariance matrix), false (do not). Default is true.

```

Output:

```

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar
                x nVar nb_cs object or double. The variance is along
                the diagonal. (Will be symmetric)

- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar
                x nVar nb_cs object or double. Along the diagonal is the
                auto-covariance/correlation with the variable itself. In
                the upper triangular part the you can find cov(x,y(-i)),
                where x is the variable along the first dimension. In
                the lower triangular part the you can find cov(x(-i),y),
                where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find
      cov(x,y(-i)) you can use
      getVariable(varargin{i}, 'y', 'x'),
      while to get cov(x(-i),y) (== cov(x,y(+i))) you
      can use getVariable(varargin{i}, 'x', 'y'). (This
      example only works if the output is of class nb_cs)

```

See also:

[nb_exprModel.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj          = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'      : The length of the bins og the domain of the density.
  Either;
    > []       : The domain will be found. See
      nb_getDensityPoints (default is that 1000
      observations of the density is stored).
    > integer   : The min and max is found but the length
      of the bins is given by the provided
      integer.
    > cell       : Must be on the format:
      {'Var1',lowerLimit1,upperLimit1,binsL1;
       'Var2',lowerLimit2,upperLimit2,binsL2;
       ...}
    - lowerLimit1 : An integer, can be nan,
      i.e. will be found.
    - upperLimit1 : An integer, can be nan,
      i.e. will be found.
    - binsL1      : An integer with the
      length of the bins. Can
      be nan. I.e. bins length
      will be adjusted to
      create a domain of 1000
      elements.

Caution : The variables not included will be given
          the default domain. No warning will be
          given if a variable is provided in the
          cell but not forecast by the model. (This
          because different models can forecast
          different variables)

Caution : The combineForecast method of the
          nb_model_group class is much faster if the
          lower limit, upper limit! Or else
          it must simulate new draws and do kernel
          density estimation for each model again
          with the shared domain of all models
          densities.

- 'startDate'   : The start date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
        the start date of the recursive forecast.
        Default is to start from the first possible
        date.

- 'endDate'     : The end date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
the ergodic steady-state for
MS-model, and in the last regime
for break-point models.

- 'startingProb' : Either a double or a char:
 - > 'zero' : Start from zero on all variables
(Except for the deterministic
exogenous variables)
 - > [] : If the model is filtered the
starting value is calculated
on filtered transition
probabilities. Otherwise the
ergodic mean is used.
 - > double : A nPeriods x nRegime or a 1 x
nRegime double. nPeriods refer to
the number of recursive periods to
forecast.
 - > 'ergodic' : Start from the ergodic transition
probabilities.
- 'stabilityTest' : Give true if you want to discard the parameter draws
that give rise to an unstable model. Default
is false.
- 'states' : Either an integer with the regime to
forecast/simulate in, or an object of class nb_ts
with the regimes to condition on. The name of the
variable must be 'states'.

Caution: For break-point models this is decided by
the dates of the breaks, and cannot be set
by this option!

- 'compareToRev' : Which revision to compare to. Default is final
revision, i.e. []. Give 5 to get fifth revision.
must be an integer. Keep in mind that this number
should be larger than the nSteps input.
- 'compareTo' : Sometime you want to evaluate the forecast of one
variable against another variable which is not
part of the model. E.g. if you use the first release
of a variable and want to compare it against the
final release. To do this you can give a cellstr
with size n x 2, where n is the number of variables
to change the the actual observations to test
against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities' : true or false. true if you want to do a
kernel density estimation of the density
forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting
them by a fitted model. Only when the 'condDB'
input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
- 'exProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.
- 'exProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.
- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_exprModel.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_exprmodel.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist ↑**

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_exprmodel.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_exprmodel.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_exprmodel.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_exprModel.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as

fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

`nb_model_generic.forecast`, `nb_model_generic.constructScore`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getResidual** ↑

`residual = getResidual(obj)`

Description:

Get residual from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- `residual` : Either a `nb_ts` or a `nb_cs` object with `residual(s)` stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualGraph** ↑

`plotter = getResidualGraph(obj)`

Description:

Get residual graph from the given estimator. The returned will be an object of class `nb_graph`, which you can call the `graphInfoStruct` method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- `obj` : An object of class `nb_model_generic` (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph`. Use the `graphInfoStruct` method or the `nb_graphInfoStructGUI` class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of `nb_model_generic` object

Input:

- `obj` : A vector of `nb_model_generic` objects

Output:

- `residualNames` : A cellstr with the unique residual names of all the models

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method `solve`.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
- 'RMSE' : One over root mean squared error
- 'MSE' : One over mean squared error
- 'MAE' : One over mean absolute error
- 'MEAN' : One over mean error
- 'STD' : One over standard error of the forecast error
- 'ESLS' : Exponential of the sum of the log scores
- 'EELS' : Exponential of the mean of the log scores
- 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.

- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_exprModel.simulatedMoments](#), [nb_exprModel.theoreticalMoments](#)
[nb_exprModel.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth SÃ¶therhagen Paulsen

► **handleMissing** ↑

ret = handleMissing(obj)

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► help ↑

```
helpText = nb_exprModel.help
helpText = nb_exprModel.help(option)
helpText = nb_exprModel.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► initialize ↑

```
obj = initialize(template)
```

Description:

Intialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.

- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_exprmodel.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [irf](#) ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsg objects.
- 'compare' : Give true if you have a scalar nb_dsg model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.
- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

```

- 'factor'           : A cell array with the factors to multiply the
                      irf of the individual model variables.
                      I.e. {'var1',100,...} or
                           {'var1','var2'},100,...}

                      If the string starts with a asterisk you will
                      multiply all the variables that contain that
                      string with the given factor.
                      E.g. {'*_GAP',100}

- 'fanPerc'         : This options sets the error band percentiles of the
                      graph, when the 'perc' input is empty. As a 1 x
                      numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
                      Default is 0.68.

- 'foundReplic'     : A struct with size 1 x replic. Each element
                      must be on the same format as obj.solution. I.e.
                      each element must be the solution of the model
                      given a set of drawn parameters. See the
                      parameterDraws method of the nb_model_generic class.

                      Caution: You still need to set 'parameterDraws'
                      to the number of draws you want to do.

- 'irfCompare'      : A struct on the same format as the irfs output from
                      this function with the IRFs to compare to.

- 'levelMethod'      : One of the following:

  > 'cumulative product'          : cumprod(1 + X,1)
  > 'cumulative sum'             : cumsum(X,1)
  > 'cumulative product (log)'    : log(cumprod(1 + X,1))
  > 'cumulative sum (exponential)' : exp(cumsum(X,1))
  > 'cumulative product (%)'      : cumprod(1 + X/100,1)
  > 'cumulative sum (%)'          : cumsum(X/100,1)
  > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
  > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
  > '4 period growth (log approx)' : nb_msum(X,3)

- 'method'            : The selected method to create error bands.
                      Default is ''. See help on the method input of the
                      nb_model_generic.parameterDraws method for more
                      this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent
                  variables. See nb_var.set_identification. Of
                  course this option only makes sense for VARs
                  identified with sign restrictions.

```

- 'normalize' : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to inf, it will normalize the max impact period.
 E.g. {'Var',1,2}, {'Var',1,inf}
 Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
 Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- 'newReplic' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- 'parallel' : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.
 Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.
 For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching models):

- > scalar : Select the the regime to take the initial transition probabilities from.
- > double : A draws x nRegime or a 1 x nRegime double. draws refer to the number set by the 'draws' input.
- > 'ergodic' : Start from the ergodic transition probabilities. Default for 'irf'.
- > 'random' : Randomize the starting values using the simulated starting points. Default for 'girf'.

- 'startingValues' : Either a double or a char:
 > double : A nVar x 1 double.
 > 'steadystate' : Start from the steady state. If you are dealing with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean

(default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

- > 'zero' : Start from zero on all variables.
- > 'random' : Randomize the starting values using the simulated starting points. Default when 'type' set to 'girf'.
- 'states' : Either an integer with the states to produce irf in, or a double with the states to condition on. Must have size periods x 1 in the last case. Applies to both MS - models and break-point models.
- 'type' : 'girf' or 'irf'. (If empty 'irf' will be used)
 - > 'girf' : Generalized impulse response function calculated as the mean difference between shocking the model with a one period shock (1 std) and no shock at all. Use the 'draws' input to set the number of simulations to use to base the calculation on. This type of irfs must be used for non-linear models.
 - > 'irf' : Standard irf for linear models. Calculated by shocking the model with a one period shock (1 std).
- 'variables' : The variables to create impulse responses of. Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.
- I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.
- Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.
- Caution: Each nb_model_generic object can have different variables and shocks.
- irfsBands : A struct with the shocks as fieldnames. Each field store

a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_exprModel.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

ret = isDensityForecast(obj)

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered ↑**

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_exprModel.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. `@(x)gt(x,0), @(x)gt(x,0)||lt(x,1)`, i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. `@(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. `{'Var1','Var1',1,@(x)gt(x,0.1)}`, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. `{'Var1','Var2',0,@(x)lt(x,0.1)}`, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.
 - > 'cov' : Same as for 'corr'.

E.g. `{'Var1','Var1',0,@(x)lt(x,0.1)}`, to test a restriction on the contemporaneous

variance.

```
> 'ss'      : A N x 2 cell, where the elements of each
               row are:
               1. The expression using any of the
                  endogenous variables of the model.
               2. Same as 4 for 'irf'.
```

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_exprModel.monteCarloFiltering](#), [nb_exprmodel.irf](#)
[nb_exprModel.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_exprModel.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is specified with some lower and upper bounds. For each draw the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.

```

- 'waitbar'      : true or false. Default is true.

- 'output'       : Either 'double' or 'logical'. Default is 'logical'.

- 'seed'         : Set the seed to use to draw the monte carlo simulated
                    parameter sets. Default is 1. Seed is set back to old
                    state after this method is called!

```

Output:

```

- paramD        : The draws made from the parameter space. As a draws x N
                  double. Use paramD(success,:) to get the accepted
                  draws.

- values         : A N x 1 logical/double. An element is true/not nan if
                  the model where solved and the test function returned
                  a value.

- solvingFailed : A N x 1 logical. An element is true if the model could
                  not be solved.

- objAcc        : A 1 x nAcc vector of nb_model_generic objects
                  representing the accepted models.

```

See also:

[function_handle](#), [nb_exprModel.mcfRestriction](#), [nb_exprmodel.solve](#)
[nb_exprModel.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **parameterDraws** ↑

```

out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)

```

Description:

Make either posterior draws or bootstrapped draws from the distribution
of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

```

- obj      : An object of a subclass of the nb_model_generic class.

- draws    : Number of draws, as an integer. Default is 1000.

- method   : The selected method to make draws.

> 'asymptotic'          : Draw from the confidence set using

```

the assumption of asymptotic normality.
 See the `nb_mlEstimator.drawParameters`
 for more on this option. Applies to models
 estimated with maximum likelihood.

> 'bootstrap' : Create artificial data to bootstrap
 the estimated parameters. Default
 for models estimated with classical
 methods.

> 'wildBootstrap' : Create artificial data by wild
 bootstrap, and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'blockBootstrap' : Create artificial data by
 non-overlapping block bootstrap,
 and then "draw" paramters based
 on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'mblockBootstrap' : Create artificial data by
 overlapping block bootstrap,
 and then "draw" paramters based
 on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'rblockBootstrap' : Create artificial data by
 overlapping random block length
 bootstrap, and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
 overlapping random block length
 bootstrap., and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'copulaBootstrap' : Uses a copula approach to draw residual
 that are autocorrelated, Does not handle
 heteroscedasticity. Only an option for
 models estimated with classical methods.

> 'posterior' : Posterior draws. Only for models
 estimated with bayesian methods.
 Default for models estimated with
 bayesian methods.

- `output` : 'param' | 'solution' | 'object'. See output section for more
 on this option.
- `stable` : Give true if you want to discard the parameter draws
 that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
 - lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
 - factors : Draws of the factors as a T x nFac x draws double.
- > 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

```
> 'object'      : Get the draws as a 1 x draws nb_model_generic object.  
                  Each element will be a model representing a given draw  
                  from the distribution of the parameters. I.e. only one  
                  output!
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```
plotter          = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the

provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF** ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

```
- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method  
or the nb_graphPagesGUI class to produce the  
graphs.  
    > 'biplot'      : A vector of nb_graph_data objects with size  
equal to the number of pairwise combination of  
the parameters that can be made. Use the  
graph method or the nb_graphMultiGUI class to  
produce the graphs.
```

See also:

[nb_exprModel.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest ↑**

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_exprModel.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_exprModel.monteCarloFiltering](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optitonal input:

- `'prior'` : Give this string as an input to compare posterior draws with the prior distributions (if available).
- `'updated'` : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- `'subplot'` : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- `'draws'` : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If `'updated'` is given as input, this will also set the the same options for the updated prior!

Output:

- `plotter` : A $1 \times \text{numCoeff}$ vector of objects of class `nb_graph_data`. Use either the `graph` method or the `nb_graphMultiGUI` class to plot the posteriors on screen.

Caution: If `'subplot'` is given it will return a scalar `nb_graph_data` object. Use the `graphSubPlots` method or the `nb_graphSubPlotGUI` class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_exprModel.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPriors ↑**

`plotter = plotPriors(obj,varargin)`

Description:

Plot priors. Only for bayesian models.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model with a set prior.

Optitonal input:

- `'subplot'` : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- `plotter` : A $1 \times \text{numCoeff}$ vector of objects of class `nb_graph_data`. Use either the `graph` method or the `nb_graphMultiGUI` class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar `nb_graph_data` object is returned. Use the `graphSubPlots` method or the `nb_graphSubPlotGUI` class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- `obj` : A vector of `nb_model_estimate` objects.

Output:

- `printed` : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_exprmodel.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

obj = removeObservations(obj, numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If

different models has different frequency this date will be converted.

- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_exprModel.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate** ↑

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also

including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.

- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.

- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.

- 'seed' : Set simulation seed. Default is 2.0719e+05.

- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

- > double : A 1 x nVar double.
- > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
- > 'steadystate' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsge models.
- > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'startingProb' : Either a double or a char:

```

> double      : A nPeriods x nRegime or a 1 x
    nRegime double. nPeriods refer to
    the number of recursive periods to
    forecast.

> 'ergodic'   : Start from the ergodic transition
    probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
- Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
 - 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
 - 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is

true.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_exprModel.graphCorrelation](#), [nb_exprModel.parameterDraws](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solutionFunc** ↑

```
yPred = nb_exprModel.solutionFunc(Z,t,E,h,beta,depTransFunc,...  
depLagFunc,exoFuncs,nLags,nExo,locDep,nDep)
```

Description:

The function that is used to forecast a model of class nb_exprModel.

Input:

- Z : A nObs x nVar double with the data of the model.
- t : The period to forecast. t <= nObs.
- E : A nRes x nSteps double with the values on the residuals.
- h : The forecasting step.
- beta : The parameters of the model as a 1 x nDep cell array. Each element contains a nExo(ii) x 1 double with the parameters of equation ii.

Output

- Z : A nObs x nVar double with the data of the model, and where the forecast is been filled in at Z(t,locDep).

See also:

[nb_exprModel.solveRecursive](#)

Written by Kenneth Sæterhagen Paulsen

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_exprModel object(s).

Input:

- obj : A vector of nb_exprModel objects.

Output:

- obj : A vector of nb_exprModel objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal ↑**

```
tempSol = nb_exprModel.solveNormal(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_exprModel.solveRecursive(results,options)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_exprmodel.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_exprModel.template()
options = nb_exprModel.template(num)
```

Description:

Construct a struct which can be provided to the nb_exprModel class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

```
- num      : Number of models to create.
```

Output:

```
- options : A struct.
```

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

```
- obj      : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
                 {variable, date, test}.

    > variable : The variable(s) to test,
                 as a string or cell array of strings.
```

```

> date      : The date as a string. If a cell array is given, a
   copy of the restriction will be made for every given
   date.

> test      : Restriction test as a function handle. The value(s)
   of the variable(s) in 'variable' at time 'date' are
   passed as arguments. Should return a logical.

```

Output:

- out : A double with the probability

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
                {'QSA_URR', 'QSA_DPQ_CP'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);

```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.

```

- alpha      : Confidence level. Default is 0.05.

- type       : Type of test:
    > '=' : Two sided test. expression == 0 (default)
            For two-sided tests it is assumed that the
            distribution is symmetric! Use
            confidence/probability intervals instead.
    > '>' : One sided test. expression > 0
    > '<' : One sided test. expression < 0

```

Output:

```

- pval      : > 'classical' : P-value of test.
                > 'bayesian' : Probability of test.

                A 1x1 double.

- ci        : A 1 x 2 double with the lower and upper bound of the
                confidence/probability interval of the tested expression.

- dist      : A nb_distribution object storing the distribution of the
                tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► **theoreticalMoments ↑**

```

[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)

```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following
way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.
Can also be a cellstr, with a subset of variables from 'full'.
- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : I the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar

`x` nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find `cov(x,y(-i))`, where `x` is the variable along the first dimension. In the lower triangular part the you can find `cov(x(-i),y)`, where `x` is the variable along the first dimension.

E.g: You have two variables `x` and `y`, then to find `cov(x,y(-i))` you can use `getVariable(varargout{i},'y','x')`, while to get `cov(x(-i),y) (== cov(x,y(+i)))` you can use `getVariable(varargout{i},'x','y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_exprModel.graphCorrelation](#), [nb_exprModel.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- `obj` : A vector of nb_model_generic objects
- `'compare2'` : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- `'method'` : One of:
 - > `'percentiles'` : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the percentiles of this distribution

```

        (default).

    > 'fev'      : Calculate the forecast error variances
                   contributed by each shock based on the
                   simulated densities of the
                   residuals/shocks of models forecast.

- 'perc'      : At which percentile you want to decompose. Must
                be scalar double. Default is 0.9. Only an options for
                'method' set to 'percentiles'.

- 'packages'  : Cell structure with groups of shocks and
                the names of the groups. If you have not
                listed a shock it will be grouped in a
                group called 'Rest'

        Must be given in the following format:

{'shock_group_name_1',..., 'shock_group_name_N';
 'shock_name_1' ,..., 'shock_name_N' }

Where 'shock_name_x' must be a string with
a shock name or a cellstr array with the
shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If
                different models has different frequency this date will
                be converted.

- 'variables' : The variables to decompose. Default is the
                dependent variables defined by the first model.

```

Output:

```

- data      : Depend on the 'method' input:

    > 'percentiles'   : A nHor x (nExo + nRes)*2 x nVars nb_ts
                        object storing the uncertainty
                        decomposition.

    > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object
                        storing the numerically calculated
                        forecast error variances/skewnesses.

- plotter : A nb_graph_ts object which the method graph can be used to
            produce graphs.

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast ↑**

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_exprmodel.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_exprmodel.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► update ↑

```
obj          = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition** ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.

- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

`nb_exprModel.parameterDraws, nb_varDecomp`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

■ **nb_favar**

Go to: [Properties](#) | [Methods](#)

A class for estimation and identification of FA-VAR models.

Superclasses:

`nb_factor_model_generic, nb_var, nb_model_generic`

Constructor:

`obj = nb_favar(varargin)`

Optional input:

- See the `set` method for more on the inputs.
(`nb_model_estimate.set`)

Output:

- `obj` : An `nb_favar` object.

See also:

`nb_model_generic, nb_model_estimate.set`

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | |
|--------------------------------|-----------------------------------|-------------------------------|-----------------------------|
| • <code>addAutoName</code> | • <code>addAutoNameIfLocal</code> | • <code>addID</code> | • <code>addIDIfLocal</code> |
| • <code>block_exogenous</code> | • <code>dependent</code> | • <code>endogenous</code> | • <code>estOptions</code> |
| • <code>exogenous</code> | • <code>factors</code> | • <code>forecastOutput</code> | • <code>name</code> |
| • <code>observables</code> | • <code>observablesFast</code> | • <code>options</code> | • <code>parameters</code> |
| • <code>reporting</code> | • <code>residuals</code> | • <code>results</code> | • <code>solution</code> |
| • <code>transformations</code> | • <code>userData</code> | | |

- **`addAutoName`** ↑

Add automatic name (first) to default name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addAutoNameIfLocal`** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **block_exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the block exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of block exogenous variables. To set it use the set function. E.g. obj = set(obj,'block_exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_VAR

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **factors** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the factors, the tex_name holds the names in the tex format. The number field holds a double with the number of factors. To set it use the set function. E.g. obj = set(obj,'factors',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_VAR

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **observables** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observables',{'Var1','Var2'}); or use the <className>.template() method.

If the model class also supports mixed frequency data, it has two additional fields; frequency and mapping. These fields stores the selected frequency and mapping for each element.

Inherited from superclass NB_FACTOR_MODEL_GENERIC

- **observablesFast** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observablesFast',...{'Var1','Var2'}); or use the <className>.template() method.

These are the variables that are included in the observation equation, but are not included in the estimation of the factors.

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsg models the residuals and exogenous property

stores the same variables (after the model being solved) !

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only) :

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only) :

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only) :

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.

- observables : A 1 x nobs cellstr with the declared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- transformations ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- userData ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- ABidentification
- appendData
- applyLongRunPriors
- assignParameters
- assignPosteriorDraws
- assignTexNames
- calculateMultipliers
- calculateStandardError
- checkModel
- checkPosteriors
- checkReporting
- conditionalTheoreticalMoments
- constructCondDB
- constructScore
- convert
- convertEach
- createVariables
- dieboldMarianoTest
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- eq
- estimate
- evalFcstAtDates
- evaluateForecast
- evaluatePrior
- explained
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getActual
- getCondDB
- getDSGEVARPriorMoments
- getDependent
- getDependentNames
- getEstimationOptions
- getEstimationStartDate
- getFactors
- getFiltered
- getForecast
- getForecastVariables
- getHistory
- getIdentification
- getIdentifiedResidual
- getLHSVars
- getModelNames
- getModelVars
- getObservables
- getOriginalVariables
- getPIT
- getParameterDrawsMethods
- getParameters
- getPosteriorDistributions

- getPredicted
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- getScore
- getVariablesList
- graphCorrelation
- handleMissing
- identDraws2Solutions
- interpretForecast
- isDensityForecast
- isIdentified
- isMixedFrequency
- isStateSpaceModel
- isbayesian
- isforecasted
- jointPredictionBands
- mcfRestriction
- monteCarloFiltering
- parameterIntervals
- plotForecast
- plotMCF
- plotMCFValues
- plotPriors
- print Cov
- priorTemplate
- set
- getPriorDistributions
- getRecursiveScore
- getResidualGraph
- getRoots
- getSolution
- grangerCausalityTest
- handleCondInfo
- help
- initialize
- irf
- isFiltered
- isMS
- isNB
- isStatic
- isestimated
- issolved
- loadPosterior
- mincerZarnowitzTest
- parameterDraws
- plotFactors
- plotForecastDensity
- plotMCFDistTest
- plotPosteriors
- print
- print _ estimation _ results
- removeObservations
- setMeasurementEqRestriction

- setPrior
 - shock_decomposition
 - simulateFromDensity
 - solve
 - solveRecursive
 - stateSpace
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
 - set_identification
 - simulate
 - simulatedMoments
 - solveNormal
 - solveVector
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
-

► **ABidentification** ↑

```
S = nb_var.ABidentification(ident,A,sigma,maxDraws,draws,ask)
```

Description:

Identification of VAR model using combination of zero and sign restrictions. See Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". Many thanks to him for supplying help!

As a extension to Binning (2013) this code also allow for magnitude restrictions.

Input:

- ident : The identification assumption, as a struct. See set_identification method of the nb_var class for more
- A : Transition matrix
- sigma : Covariance matrix of the VAR residuals.
- maxDraws : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.
- draws : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Output:

- S : A nb_struct with field W, with the stored map from structural shocks to dependent variables.

Written by Kenneth Sæterhagen Paulsen,
Subfunctions of this code are written by Andrew Binning.

Inherited from superclass NB_VAR

► **appendData** ↑

obj = appendData(obj, DB)

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **applyLongRunPriors** ↑

```
obj = applyLongRunPriors(obj,H,phi)
```

Description:

Apply priors for the long run as in Giannone et. al (2014).

If the prior field of the options property is not yet assign a prior, a jeffrey prior is used as default.

Caution: Please do not call this method before nb_var.setPrior!

Input:

- obj : A N x M matrix of nb_var objects, with nDep dependent variables.

- phi : Either a nDep x 1 double vector with the prior shrinkage parameter for each equation, or a m x 2 cell. m <= nDep. The format of the cell input must be as follows;

```
{'var1',1;'var2',0.5}
```

If some dependent variables are not assign any parameter the default is 1.

- H : A q x nDep matrix, where q <= nDep. Each row is the cointegration vector to apply on the prior. If q < nDep the null space will be applied to the rest of the rows.

Or it can be a q x 1 cellstr with the cointegration relations. Again q <= nDep, and if q < nDep the null space will be applied to the rest of the rows. Please note that only dependent variables may be applied in the cointegration relations! Example:

```
{'var1 - var2','var3 + var1'}
```

Output:

- obj : A N x M matrix of nb_var objects.

See also:

[nb_favar.set](#), [nb_favar.setPrior](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model have been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames** ↑

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj, varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3, 0.5, 0.7, 0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_favar.inf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

obj = calculateStandardError(obj,method,draws)

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.

- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterior** ↑

```
[DB,plotter,pAutocorr] = checkPosterior(obj,varargin)
```

Description:

Plot posteriors draws in the order they were drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the

all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.

Caution : Posterior draws is already made at the estimation stage.

- 'nSteps' : Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB           = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  
horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
           allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **convert ↑**

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables ↑**

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

```
> second column : Any expression that can be interpreted
by the nb_ts.createVariable method.
```

```

> third column : Any expression that can be interpreted
    by the nb_ts.createShift method.

> fourth column : Comments

- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is
    important that this option is higher than the number of
    forecasting/irf steps, or else the output will be nan!

```

Output:

- obj : The NxM object itself added the model variables. The
 expressions input is stored in the property
 transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending.
 Use the graphInfoStruct method or the nb_graphInfoStruct
 class to produce the graph.

Examples:

```

expressions = {%
  Name,      input,    shift,    description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};
model = model.createVariables(expressions)

```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth SÃ¶tterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest ↑**

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band width selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_favar.uncondForecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_favar.forecast](#), [nb_favar.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_favar.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_favar.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic
- Optional inputs;
- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
 - 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
 - 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
 - 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargin{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you

can use `getVariable(varargin{i},'x','y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_favar.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [eq](#) ↑

`ret = eq(obj,other)`

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- `obj` : A nb_model_generic object either with same size as others, or equal to 1.
- `others` : A nb_model_generic model with `dim == 3` or less

Output:

- `ret` : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind     = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [estimate](#) ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d,{'MSE'},0,1)
m = evalFcstAtDates(b,d,{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

```
- prior : See options.prior.  
- par : A nEst x 1 double with the value of the point of evaluation.
```

Output:

```
- logPriorD: The log prior density at the evaluated point.
```

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [explained ↑](#)

```
[expl,plotter] = explained(obj)
```

Description:

Get how much of the variation in the data the factors explain.

Caution: Remember that for dynamic factor models the factors are not orthogonal.

Input:

```
- obj : A scalar nb_factor_model_generic object.
```

Output:

```
- expl : A nb_cs object with size nFactors x 1.  
- plotter : A nb_graph_cs object. Use the graph method or the  
nb_graphPagesGUI class to plot it on the screen.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► [forecast ↑](#)

```
obj = forecast(obj,nSteps,varargin)  
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
- 'varOfInterest' : A char with the variable to produce the forecast of.

Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!

- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution:** 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length of the bins is given by the provided integer.

```

> cell      : Must be on the format:

{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...
}

- lowerLimit1 : An integer, can be nan,
  i.e. will be found.

- upperLimit1 : An integer, can be nan,
  i.e. will be found.

- binsL1      : An integer with the
  length of the bins. Can
  be nan. I.e. bins length
  will be adjusted to
  create a domain of 1000
  elements.

Caution : The variables not included will be given
the default domain. No warning will be
given if a variable is provided in the
cell but not forecast by the model. (This
because different models can forecast
different variables)

Caution : The combineForecast method of the
nb_model_group class is much faster if the
lower limit, upper limit! Or else
it must simulate new draws and do kernel
density estimation for each model again
with the shared domain of all models
densities.

- 'startDate'   : The start date of forecast. Must be a string or an
object of class nb_date. If empty the estim_end_date
+ 1 will be used.

Caution: If 'fcstEval' is not empty this will set
the start date of the recursive forecast.
Default is to start from the first possible
date.

- 'endDate'     : The end date of forecast. Must be a string or an
object of class nb_date. If empty the estim_end_date
+ 1 will be used.

Caution: If 'fcstEval' is empty this input will
have nothing to say.

- 'saveToFile'   : Logical. Save densities and domains to files. One
file for each model. Default is false (do not).

- 'observables'  : A cellstr with the observables you want the
forecast of. Only for factor models. Will be
discared for all other types of models.

- 'condDB'       : One of the following:

```

> A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.

> A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.

> A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)

> A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with

uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.

- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
 - If not provided. Model stds are used.
 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!
- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogneous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:
 This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'startingProb' : Either a double or a char:

```

> [ ]           : If the model is filtered the
                  starting value is calculated
                  on filtered transition
                  probabilities. Otherwise the
                  ergodic mean is used.

> double       : A nPeriods x nRegime or a 1 x
                  nRegime double. nPeriods refer to
                  the number of recursive periods to
                  forecast.

> 'ergodic'    : Start from the ergodic transition
                  probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws
                    that give rise to an unstable model. Default
                    is false.

- 'states'      : Either an integer with the regime to
                    forecast/simulate in, or an object of class nb_ts
                    with the regimes to condition on. The name of the
                    variable must be 'states'.

Caution: For break-point models this is decided by
          the dates of the breaks, and cannot be set
          by this option!

- 'compareToRev' : Which revision to compare to. Default is final
                   revision, i.e. []. Give 5 to get fifth revision.
                   must be an integer. Keep in mind that this number
                   should be larger than the nSteps input.

- 'compareTo'   : Sometime you want to evaluate the forecast of one
                   variable against another variable which is not
                   part of the model. E.g. if you use the first release
                   of a variable and want to compare it against the
                   final release. To do this you can give a cellstr
                   with size n x 2, where n is the number of variables
                   to change the the actual observations to test
                   against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
                      kernel density estimation of the density
                      forecast.

- 'exoProj'      : Tolerate missing exogenous variables by projecting
                   them by a fitted model. Only when the 'condDB'
                   input is empty!

Options are:

- ''   : No projection are done. Error will be
         given if some exogenous variables are
         not given any conditional information.
         Default.

- 'ar' : The exogenous variables will be fitted by
         a univaiate AR model with the lag length

```

chosen with a 'aicc' criterion.

Caution : If density forecast are produced
the AR estimates will be
bootstrapped, and the exogenous
variables will be re-sampled
based on the bootstrapped values.

Caution : If forecast are being produced
recursively with recursively
estimated parameters, the
forecast/bootstrapping of the
exogenous variables will be done
based on recursively estimated AR
coefficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.
- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummy variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the

variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
 - 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so

in this case this output will be true for all models.

See also:

[nb_favar.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_favar.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist ↑**

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_favar.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getActual ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.

- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.

- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_favar.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by `nb_var.priorTemplate('dsge')`.

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

dependentNames = getDependentNames(obj)

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

outOpt = getEstimationOptions(obj)

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

start = getEstimationStartDate(obj)

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFactors** ↑

factors = getFactors(obj)

Description:

```
Get factors from a estimated nb_factor_model_generic object
```

Input:

- obj : An object of class nb_factor_model_generic

Output:

- factors : A nb_ts object with the estimated factors of the model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **getFiltered ↑**

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_favar.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```

fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date'      : A string or a nb_date object with the date of the
                  wanted forecast. E.g. '2012Q1'. The fcstData will
                  be a nb_ts object with size nHor x nVar x nPerc + 1,
                  while fcstPercData will be empty. nPerc will then be
                  the number of percentiles/simualtions. Mean will be at
                  last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                  nPeriods + nHor x nVar x nHor nb_ts object. This
                  option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
                options for the 'date' and 'horizon' outputType.
                Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                  size nRec x nVars. While the fcstPercData output
                  will be a nb_ts object with size nRec x nVars x nSim.
                  You can set the horizon to return by the optional
                  input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_favar.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables** ↑

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getIdentification ↑

```
matrix = getIdentification(obj)
```

Description:

Get restrictions on the structural shock matrix, i.e C in the equation below:

$y(t) = A*y(t-1) + B*x(t) + C*e(t)$, where $e(t) \sim N(0, I)$.

Input:

- obj : A identified scalar nb_var object.

Output:

- matrix : nEndo x nEndo cell array.

See also:

[nb_favar.set_identification](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► getIdentifiedResidual ↑

```
residual = getIdentifiedResidual(obj)
```

Description:

Get identified residual from a estimated nb_var object

If we have a VAR it can be written as (dropping exogenous variables):

$y_t = A*y_{t-1} + e_t$

A identified VAR can be written as:

$y_t = A*y_{t-1} + C*d_t$

Therefore the identified residuals/shocks can be found as:
d_t = inv(C)*e_t

Caution: Will only return the residual that uses the full sample. I.e.
when recursive estimation is done, only the residuals from the
last estimation will be returned.

Input:

- obj : An object of class nb_var

Output:

- residual : A nb_ts object with the identified residual(s) stored.
As a nobs x nEq x nIdent. Where nIdent is the number of
identified C matrices, i.e. when the user have asked to
identify more than one matrix (underidentification).

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **getLHSVars ↑**

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.

- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getObservables** ↑

```
observables = getObservables(obj)
```

Description:

Get observables from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- observables : A nb_ts object with observables of the factor model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.
- Optional input
 - varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).
 - : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
 - : Give 'double' to return the value as a numerical array.
 - : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

parameters

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions** ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPriorDistributions** ↑

[distr,paramNames] = getPriorDistributions(obj)

Description:

Get prior distributions of B-VAR model.

Input:

- obj : An object of class nb_var.

Output:

- distr : A 1 x numCoeff nb_distribution object.

- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **getRecursiveEstimationGraph** ↑

plotter = getRecursiveEstimationGraph(obj)

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph_ts`. Use the `graphSubPlots` method or the `nb_subplotGUI` class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- `obj` : A vector of `nb_model_forecast` objects.
- `type` : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- `dim` : Either 'variables' to get the recursive scores of all the variables forecast by a model (the `obj` input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- `startDate` : The date to begin constructing the score. Can be empty. Either as a string or an object of class `nb_date`.
- `endDate` : The date to end constructiong the score. Can be empty. Either as a string or an object of class `nb_date`.
- `invert` : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

residual = getResidual(obj)

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

plotter = getResidualGraph(obj)

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores

- 'MLS' : Mean log score

If the object intput is a vector, the type input can also be a cellstr array wit matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursivly, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList ↑**

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **grangerCausalityTest ↑**

```
[test,pval,results] = grangerCausalityTest(obj,precision)
```

Description:

Test for granger causality between variables using a joint F-test that all the parameters of the lags of a variable y is zero in the equation where x is the dependent variable. If the hypothesis can be rejected we say that y granger cause x.

Caution: If recursive estimation is done it will anyway only do the test using the parameters estimated over the full sample.

Input:

- obj : A 1 x 1 nb_var object
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : The test statistics. As a nDep x nDep double. The diagonal will return nan.
- pval : The p-value of the test statistics. As a nDep x nDep double. The diagonal will return nan.
- results : A string with the printed results of the test.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_VAR

► graphCorrelation ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against emprirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from

`nb_model_generic.simulatedMoments` (SIM) or
`nb_model_generic.theoreticalMoments` (THEO), the second input must be
the c output from `nb_model_generic.empiricalMoments` (EMP), the third
input can be the ac1 output from SIM or THEO, the fourth input can be
the ac1 output from EMP, and so on.

Caution : The inputs must be given as `nb_cs` objects.

Caution : Inputs must come in pairs.

Output:

- `plotter` : An object of class `nb_graph_cs`. Use the `graph` method or the `nb_graphPagesGUI` class.

See also:

`nb_favar.simulatedMoments`, `nb_favar.theoreticalMoments`
`nb_favar.empiricalMoments`, `nb_graphPagesGUI`, `nb_graph_cs`

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **handleCondInfo** ↑

`ret = handleCondInfo(obj)`

Description:

Check if a `nb_model_estimate` object handle conditional info.

Input:

- `obj` : A scalar `nb_model_estimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the
model handle conditional info.

Written by Kenneth SÃ¶terhagen Paulsen

► **handleMissing** ↑

`ret = handleMissing(obj)`

Description:

```
Check if a nb_model_estimate object handle missing observations.
```

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_favar.help  
helpText = nb_favar.help(option)  
helpText = nb_favar.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **identDraws2Solutions** ↑

```
sol = identDraws2Solutions(obj)
```

Description:

This function assumes that you have used the set_identification function with the input 'type' set to 'combination'.

Input:

- obj : An object of class nb_var.

Output:

- sol : A struct storing the solution of the model. Same format as the solution property of the nb_var class.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_VAR

► initialize ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.

- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_favar.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

[irfs, irfsBands, plotter] = irf(obj, varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects
Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against each other. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the

nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or
{{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:
 - > 'cumulative product' : cumprod(1 + X,1)

```

> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
    the map from structural shocks to dependent
    variables. See nb_var.set_identification. Of
    course this option only makes sense for VARs
    identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

- 'parallelL' : Give true if you want to do the irfs in parallel.

```

This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.

- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for `nb_dsg` objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method `nb_var.set_identification` for more.

- 'settings' : Extra graphs settings given to `nb_plots` function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

```

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar      : Select the the regime to take the
                     initial transition probabilities
                     from.

    > double      : A draws x nRegime or a 1 x
                     nRegime double. draws refer to
                     the number set by the 'draws'
                     input.

    > 'ergodic'   : Start from the ergodic transition
                     probabilities. Default for 'irf'.

    > 'random'    : Randomize the starting values
                     using the simulated starting
                     points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double      : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing
                      with a MS-model or a break-point model you can
                      indicate the regime by 'steadystate(regime)'. E.g.
                      to start in regime 1 give; 'steadystate(1)'. For
                      MS - models the default is the ergodic mean
                      (default as long as 'girf' is not selected as
                      'type'), while for break-point models it is to
                      start from the first regime.

    > 'zero'       : Start from zero on all variables.

    > 'random'    : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'       : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'         : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'  : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'    : The variables to create impulse responses of.
                     Default is the dependent variables defined by the

```

first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_favar.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isIdentified** ↑

```
ret = isIdentified(obj)
```

Description:

Is the nb_var object identified or not.

Input:

- obj : A nb_var object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **isMS** ↑

ret = isMS(obj)

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

ret = isStateSpaceModel(obj)

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted ↑**

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved ↑**

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.
- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the

dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_favar.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. @(x)gt(x,0), @(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. @(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1), i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.
 - > 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.
 - > 'ss' : A N x 2 cell, where the elements of each row are:
 1. The expression using any of the endogenous variables of the model.
 2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_favar.monteCarloFiltering](#), [nb_favar.irf](#)
[nb_favar.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [mincerZarnowitzTest](#) ↑

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_favar.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► monteCarloFiltering ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

function_handle, nb_favar.mcfRestriction, nb_favar.solve
nb_favar.assignParameters

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.

```

> 'wildBootstrap'      : Create artificial data by wild
                           bootstrap, and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'blockBootstrap'    : Create artificial data by
                           non-overlapping block bootstrap,
                           and then "draw" paramters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'mblockBootstrap'   : Create artificial data by
                           overlapping block bootstrap,
                           and then "draw" paramters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'rblockBootstrap'   : Create artificial data by
                           overlapping random block length
                           bootstrap, and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'wildBlockBootstrap': Create artificial data by wild
                           overlapping random block length
                           bootstrap., and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'copulaBootstrap'   : Uses a copula approach to draw residual
                           that are autocorrelated, Does not handle
                           heteroscedasticity. Only an option for
                           models estimated with classical methods.

> 'posterior'         : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores
 available.
- 'newDraws' : When out of parameter draws, this factor decides how

many new draws are going to be made. Default is 0.1.
I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.

- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
> 'param' : Default. A struct with the following fields:
 beta : A nPar x nEq x draws double with the estimated parameters.
 sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:
 lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
 factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotFactors ↑

```
plotter = plotFactors(obj,errBands)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic
- errBands : Give true to add one std bands around the factor estimates.
Default is false.

Output:

- plotter : A nb_graph_ts object to use to plot the factors. Use the graphSubPlots method or the nb_graphSubPlotGUI.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **plotForecast** ↑

```
plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forcast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF** ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.

- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_favar.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest ↑**

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo

filtering.

- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_favar.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues ↑**

plotter = nb_model_generic.plotMCFValues(paramD, params, nameValue)

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_favar.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optitonal input:

- `'prior'` : Give this string as an input to compare posterior draws with the prior distributions (if available).
- `'updated'` : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- `'subplot'` : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- `'draws'` : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If `'updated'` is given as input, this will also set the the same options for the updated prior!

Output:

- `plotter` : A $1 \times \text{numCoeff}$ vector of objects of class `nb_graph_data`. Use either the `graph` method or the `nb_graphMultiGUI` class to plot the posteriors on screen.

Caution: If `'subplot'` is given it will return a scalar `nb_graph_data` object. Use the `graphSubPlots` method or the `nb_graphSubPlotGUI` class to plot the posteriors on screen.

See also:

`nb_graph_data`, `nb_graphMultiGUI`, `nb_graphSubPlotGUI`
`nb_favar.getPosteriorDistributions`

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPriors** ↑

```
plotter = plotPriors(obj,varargin)
```

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov ↑**

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_favar.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **priorTemplate ↑**

```
prior = nb_favar.priorTemplate()  
prior = nb_favar.priorTemplate(type)  
prior = nb_favar.priorTemplate(type, num)
```

Description:

Not yet finished!!

Construct a struct which can be given to the method setPrior.

The structure provided the user the possibility to set different prior options.

Input:

- type : A string;
- num : Number of prior templates to make.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **removeObservations ↑**

```
obj = removeObservations(obj, numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setMeasurementEqRestriction** ↑

```
obj = setMeasurementEqRestriction(obj, restrictions)
```

Description:

Add restrictions in the measurement equation of the model on the form;

```
restricted = parameters*variables'
```

e.g.

```
X = [0.5,0.5]*[Y;Z]
```

where X is a observed variable, while Y and Z are state variables.
So for a mixed frequency var, X and Y must be included in the state
equation. An error is provided if this is not the case!

For the xxxMF priors this information is taken into account, otherwise
it will only be used during conditional forecasting, i.e. the measurement
restrictions will be appended to the solution after estimation is done.

See [nb_var.priorTemplate](#) for more on the supported priors.

Input:

- obj : An object of class nb_var
- restrictions : A struct array with fields
 - > 'restricted' : A one line char with the variable to restrict. It must be a dependent variable or block exogenous variable.
 - > 'parameters' : A cellstr with the names of the variables storing the parameters of the restriction, or a double array. Must have same length as 'variables'.
 - > 'variables' : A cellstr with the names of the variables included in the restriction.
 - > 'frequency' : Set to the frequency of the observables. This is only important for the mixed frequency VAR model. Give [], if the restriction is on the same frequency as the data. Either 1 (yearly), 4 (quarterly) or 12 (monthly).
 - > 'mapping' : Sets the mapping to use for this restricted variable. See the method [nb_mfvar.setMapping](#) for more on the

mappings to choose from.
> 'R_scale' : Inverse prior scale of the variance
of the measurement error of this
restrictions.

The variance of the measurement error
is constructed by dividing the
variance of each 'restricted' variable
by this scaling parameter.

Output:

- obj : An object of class nb_var where the measurement equation restrictions are added.

See also:

[nb_favar.template](#), [nb_bVarEstimator.applyMeasurementEqRestriction](#)
[nb_favar.priorTemplate](#), [nb_mfvar.setMapping](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **setPrior** ↑

obj = setPrior(obj,prior)

Description:

Set priors of a vector of nb_var objects. The prior input must either be a struct or nb_struct with same size as obj or have size 1.

The provided struct will be added to the property options as the field prior.

See [nb_var.priorTemplate](#) for the format of the struct.

Caution : The estim_method field of the option property will automatically be set to 'bvar'.

Caution : Not yet supported for the subclass nb_favar!

Input:

- obj : A vector of nb_var objects
- prior : A struct or nb_struct with either matching numel of obj or size 1. If size is 1 all models gets the same prior settings.

Output:

- obj : A vector of nb_var objects with the priors set.

See also:

[nb_favar.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **set_identification** ↑

```
obj = set_identification(obj,type,varargin)
```

Description:

Identify a VAR model. Either using cholesky restrictions or combinations of sign and zero restrictions (short, mid and long term).

The method solve must be called after adding the identifying restrictions

See Andrew Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". This code is an adaption of his code to the NBToolbox, except that also magnitude restriction is possible using this code.

Input:

- obj : A vector of nb_var objects, or subclasses of this class.
- type : Either 'cholesky' or 'combination'
- varargin (Optional inputs) :
 - > 'cholesky' :
 - 'ordering' : A cellstr with the ordering of the identification. Default is to use the ordering of the dependent variables of the model. The ordering is so that the first variable is the variable that is only influenced by one shock in period 1, the second variable is the variable that is influenced by two shock in period 1, and so on.
 - Caution : All variables of the VAR must be included in this option.
 - > 'combination' :
 - 'maxDraws' : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.
 - 'draws' : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Caution : When it comes to producing IRFs with sign restrictions you have two options. The first is to set 'draws' to a large number, say 1000, and then produce irf for each of those draws. The second approach is to draw parameters using bootstrap, MC methods or from the posterior, and then to make one draw of the C for each draw of the paramters.

```

- 'restrictions' : A nRestriction x 4 cell;
  {Dep,Shock,period,type,value;...}

  > Dep      : The dependent variable to add the restriction to. As a string.

  > Shock    : The name of the identified structural shock. As a string.

  > period   : The period of the restriction, as a double. For on impact set 0 and for cumulative restriction set it to inf.

  Use s:e to add a restriction on the cumulative contribution of a shock to a variable starting at s and ending at e. E.g. 0:10.

  > type     : Either 0 for zero restriction, '+/' '-' for sign restrictions or '>'/'<' for magnitude restrictions.

  Or 'none' : If you want to add a structural shock with no restriction, as you already have exactly identified N-1 shocks.

  > value    : A 1x1 double with the value of the magnitude restriction.
  
```

Output:

- obj : Identified nb_var objects. See the property field identification of the property solution.

Examples:

See ...\\Econometrics\\test\\test_nb_var

Written by Kenneth Sæterhagen Paulsen
Subfunctions are written by Andrew Binning 2013

Inherited from superclass NB_VAR

► shock_decomposition ↑

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';  
'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more

this input. An extra option is:

```
> 'identDraws' : Uses the draws of the matrix with  
      the map from structural shocks to dependent variables.  
See nb_var.set_identification. Of course this option  
only makes sense for VARs identified with sign  
restrictions.
```

- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.

- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.

- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.

- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.

- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.

- 'model2' : A struct with the solution of the second model to use for decomposition.

- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

`nb_favar.parameterDraws, nb_shockDecomp`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **simulate** ↑

`out = simulate(obj,nSteps,varargin)`

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `nSteps` : Number of simulation steps. As a `1x1 double`. Default is 100.

Optional inputs:

- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- `'bounds'` : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - `'shock'` : Name of the shock to match the restriction on the bounds of the given variable.
 - `'lower'` : The lower bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.
 - `'upper'` : The upper bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps double` matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a `1x1 double` or a `nSteps x 1 double`.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' :
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty

the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadyState' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsg models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consists of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the

'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i},'y','x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i},'x','y'). (This example only works if the output is of class nb_cs)

See also:

[nb_favar.graphCorrelation](#), [nb_favar.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_favar object(s).

Input:

- obj : A vector of nb_favar objects.

Output:

- obj : A vector of nb_favar objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_favar.solveNormal(results,opt,ident)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_favar.solveRecursive(results,opt,ident)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **stateSpace** ↑

```
[x0,P0,d,H,R,T,c,A,B,Q,G,obs,failed] = nb_var.stateSpace(par,nDep,...  
nLags,nExo,restr,restrVal,stabilityTest)
```

Description:

Returns a state-space representation of an VARX(nLags) model.

Observation equation:

y = d + Hx + Tz + v

State equation:

x = c + Ax_1 + Gz + Bu

Input:

- par : The parameter vector of the VARX model.
 - Order:
 - constant
 - exogenous
 - lagged endogenous
 - residual std
- nDep : The number of dependent variables of the model.
- nLags : the number of lags of the VAR.
- constant : Include constant or not. true or false.
- nExo : Number of exogenous variables included in the model.
- restr : A nDep x nDep*nLags logical matrix. true for the non-restricted parameters.
- restrVal : Values of the restricted parameters.
- stabilityTest : Check stability of system. Sets failed to true if model is not stationary. Default is false.

Output:

- x0 : Initial state vector.
- P0 : Initial variance.
- See equation above
- obs : Index of observables in the state vector.
- failed : See stabilityTest input.

Examples:**See also:**

[nb_kalmanlikelihood](#), [nb_kalmanlikelihood_missing](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► struct ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_favar.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_favar.template()
options = nb_favar.template(num)
```

Description:

Construct a struct which can be provided to the nb_favar class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
- > variable : The variable(s) to test,
as a string or cell array of strings.
- > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
- > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or
posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as

```

variables. See model.parameters.name.

- method      : A string with the method to use. For bootstrap method see
                nb_bootstrap. For bayesian models 'posterior' is the only
                option. Default is 'bootstrap' for classical models, while
                'posterior' is the default for bayesian models.

- draws       : Number of draws from the parameter distribution. Default
                is 1000.

- alpha       : Confidence level. Default is 0.05.

- type        : Type of test:
                > '=' : Two sided test. expression == 0 (default)
                        For two-sided tests it is assumed that the
                        distribution is symmetric! Use
                        confidenc/probabillty intervals instead.
                > '>' : One sided test. expression > 0
                > '<' : One sided test. expression < 0

```

Output:

```

- pval        : > 'classical' : P-value of test.
                > 'bayesian'   : Probabilty of test.

                A 1x1 double.

- ci          : A 1 x 2 double with the lower and upper bound of the
                confidence/probabillty interval of the tested expression.

- dist        : A nb_distribution object storing the distribution of the
                tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► **theoreticalMoments** ↑

```
[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargout{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_favar.graphCorrelation](#), [nb_favar.parameterDraws](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► uncertaintyDecomposition ↑

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of

interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).

- > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_favar.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_favar.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update** ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve
> 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► variance_decomposition ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with

the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.

- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.
- Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.

- `plotter` : A $1 \times nModel$ vector of `nb_graph_cs` objects. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
- `plotterBands`: A $1 \times nModel$ struct. Each field is a $1 \times nVars$ vector of `nb_graph_cs` objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_favar.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

■ `nb_fm`

Go to: [Properties](#) | [Methods](#)

A class for estimation single equation factor model.

```
y_t = beta*F_t + beta0*X_t + e_t
```

Superclasses:

`nb_factor_model_generic`, `nb_model_generic`

Constructor:

```
obj = nb_fm(varargin)
```

Optional input:

- See the `set` method for more on the inputs.
(`nb_model_estimate.set`)

Output:

- `obj` : An `nb_fm` object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- | | | | |
|-------------------------------|-----------------------------------|----------------------------|-----------------------------|
| • <code>addAutoName</code> | • <code>addAutoNameIfLocal</code> | • <code>addID</code> | • <code>addIDIfLocal</code> |
| • <code>dependent</code> | • <code>endogenous</code> | • <code>estOptions</code> | • <code>exogenous</code> |
| • <code>forecastOutput</code> | • <code>name</code> | • <code>observables</code> | • <code>options</code> |
| • <code>parameters</code> | • <code>reporting</code> | • <code>residuals</code> | • <code>results</code> |
| • <code>solution</code> | • <code>transformations</code> | • <code>userData</code> | |

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **observables** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observables',{'Var1','Var2'}); or use the <className>.template() method.

If the model class also supports mixed frequency data, it has two additional fields; frequency and mapping. These fields stores the selected frequency and mapping for each element.

Inherited from superclass NB_FACTOR_MODEL_GENERIC

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.

- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFcstAtDates
- evaluatePrior
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFactors
- getForecast
- getHistory
- getModelNames
- getObservables
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- explained
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual

- getResidualGraph
 - getRoots
 - getSolution
 - graphCorrelation
 - handleMissing
 - initialize
 - irf
 - isFiltered
 - isMixedFrequency
 - isStateSpaceModel
 - isbayesian
 - isforecasted
 - jointPredictionBands
 - mcfRestriction
 - monteCarloFiltering
 - parameterIntervals
 - plotForecast
 - plotMCF
 - plotMCFValues
 - plotPriors
 - printCov
 - removeObservations
 - shock_decomposition
 - simulateFromDensity
 - solve
 - solveRecursive
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
 - getResidualNames
 - getScore
 - getVariablesList
 - handleCondInfo
 - help
 - interpretForecast
 - isDensityForecast
 - isMS
 - isNB
 - isStatic
 - isestimated
 - issolved
 - loadPosterior
 - mincerZarnowitzTest
 - parameterDraws
 - plotFactors
 - plotForecastDensity
 - plotMCFDistTest
 - plotPosteriors
 - print
 - print_estimation_results
 - set
 - simulate
 - simulatedMoments
 - solveNormal
 - solveVector
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
-

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters** ↑

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_fm.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fm.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_fm.forecast](#), [nb_fm.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_fm.forecast, nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast, nb_fm.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

```

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables.
            'dependent' will return the moment of the dependent
            variables. 'dependent' is default.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or
               not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to
             true.

- 'type' : Either 'covariance' or 'correlation'. Default is
            'correlation'.

- 'startDate' : The start date of the calculations. Default is the start
                date of the options.data property. Only for time-series!
                Must be a string or a nb_date object.

- 'endDate' : The end date of the calculations. Default is the end
               date of the options.data property. Only for time-series!
               Must be a string or a nb_date object.

- 'demean' : true (demean data during estimation of the
              autocovariance matrix), false (do not). Default is true.

```

Output:

```

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar
                x nVar nb_cs object or double. The variance is along
                the diagonal. (Will be symmetric)

- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar
                x nVar nb_cs object or double. Along the diagonal is the
                auto-covariance/correlation with the variable itself. In
                the upper triangular part you can find cov(x,y(-i)),
                where x is the variable along the first dimension. In
                the lower triangular part you can find cov(x(-i),y),
                where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find
      cov(x,y(-i)) you can use
      getVariable(varargin{i}, 'y', 'x'),
      while to get cov(x(-i),y) (== cov(x,y(+i))) you
      can use getVariable(varargin{i}, 'x', 'y'). (This
      example only works if the output is of class nb_cs)

```

See also:

[nb_fm.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj          = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **explained** ↑

```
[expl,plotter] = explained(obj)
```

Description:

Get how much of the variation in the data the factors explain.

Caution: Remember that for dynamic factor models the factors are not orthogonal.

Input:

- obj : A scalar nb_factor_model_generic object.

Output:

- expl : A nb_cs object with size nFactors x 1.
- plotter : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to plot it on the screen.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
 - 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
 - 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
 - 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
- Only for density forecast:
- 'logScore' : Log score
- Can also be a cellstr with the above listed evaluation types.
- Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.
- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density

forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.

- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length of the bins is given by the provided integer.
 - > cell : Must be on the format:
`{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...}`
 - lowerLimit1 : An integer, can be nan, i.e. will be found.
 - upperLimit1 : An integer, can be nan, i.e. will be found.
 - binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given

the default domain. No warning will be given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The `combineForecast` method of the `nb_model_group` class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- `'startDate'`
 - : The start date of forecast. Must be a string or an object of class `nb_date`. If empty the `estim_end_date + 1` will be used.
 - Caution:** If '`fcstEval`' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.
- `'endDate'`
 - : The end date of forecast. Must be a string or an object of class `nb_date`. If empty the `estim_end_date + 1` will be used.
 - Caution:** If '`fcstEval`' is empty this input will have nothing to say.
- `'saveToFile'`
 - : Logical. Save densities and domains to files. One file for each model. Default is `false` (do not).
- `'observables'`
 - : A `cellstr` with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- `'condDB'`
 - : One of the following:
 - > A `nb_ts` object with size `nSteps x nCondVar` with the information to condition on.
If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A `nb_data` object with size `nSteps x nCondVar x nPeriods` with the information to condition on. The `dataNames` property must be set to the start dates of the recursive conditional information. `nPeriods` will be the number of recursive periods.
 - > A `nb_ts` object with size `nSteps x nCondVar x 3` with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given

as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)

> A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual

to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.

> StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

> Horizon : Anticipated horizon of the shocks/residual. 1 is default.

> Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned

included the lag variables. Also including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Including exogenous and shocks.

- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.

> double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.

> 'mean' : Start from the mean of the data observations. Default.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

> 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'startingProb' : Either a double or a char:

> [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'states' : Either an integer with the regime to forecast/simulate in, or an object of class nb_ts

with the regimes to condition on. The name of the variable must be 'states'.

Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!

- 'compareToRev' : Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
- 'compareTo' : Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
 - Options are:
 - '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
 - 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.
- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the

method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matix. For the order of the

variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_fm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fm.forecast](#)

Written by Per Bjarne Bye, Kenneth Åkerhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► forecastPerc2ParamDist ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fm.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditinal information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_fm.forecast](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- `obj` : An object of class nb_model_generic.
- `dsgeVar` : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- `'maxIter'` : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- `'tol'` : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- `GammaYY` : A $N \times N$ double with the nonstandardized sample moments of the left-hand variables of the VAR.
- `GammaXX` : A $(C + N*L) \times (C + N*L)$ double with the nonstandardized sample moments of the right-hand variables of the VAR.
- `GammaXY` : A $(C + N*L) \times N$ double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

dependentNames = getDependentNames(obj)

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)

Written by Kenneth SÃ¶terhagen Paulsen
```

► **getEstimationStartDate ↑**

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFactors ↑**

```
factors = getFactors(obj)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- factors : A nb_ts object with the estimated factors of the model stored.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **getFiltered ↑**

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_fm.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0, -1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is

returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!

Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_fm.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getHistory ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getLHSVars ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj,varsIn)
```

Description:

Get all left hand side variables of the model.
For factor models the observable (+ observableFast) are added as well.
The list will be sorted.
Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getObservables ↑

```
observables = getObservables(obj)
```

Description:

Get observables from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- observables : A nb_ts object with observables of the factor model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► getOriginalVariables ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► getPIT ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option

is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

: Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

```
- obj : An object of class nb_model_generic.
```

Optitonal input:

```
- 'draws' : The number of draws to sample from the posterior to base  
the kernel estimation on, if empty all draws are used for  
estimation.
```

Output:

```
- distr : A 1 x numCoeff nb_distribution object.
```

```
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- residual : Either a nb_ts or a nb_cs object with predicted variable(s)  
stored.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_subplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

residual = getResidual(obj)

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

plotter = getResidualGraph(obj)

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames ↑**

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,... 
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_fm.simulatedMoments](#), [nb_fm.theoreticalMoments](#)
[nb_fm.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help** ↑

```
helpText = nb_fm.help  
helpText = nb_fm.help(option)  
helpText = nb_fm.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.

- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_fm.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

[irfs, irfsBands, plotter] = irf(obj, varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the

nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

> 'cumulative product'	: cumprod(1 + X,1)
------------------------	--------------------

```

> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
    the map from structural shocks to dependent
    variables. See nb_var.set_identification. Of
    course this option only makes sense for VARs
    identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

- 'parallelL' : Give true if you want to do the irfs in parallel.

```

This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.

- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for `nb_dsg` objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method `nb_var.set_identification` for more.

- 'settings' : Extra graphs settings given to `nb_plots` function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

```

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar      : Select the the regime to take the
                     initial transition probabilities
                     from.

    > double      : A draws x nRegime or a 1 x
                     nRegime double. draws refer to
                     the number set by the 'draws'
                     input.

    > 'ergodic'   : Start from the ergodic transition
                     probabilities. Default for 'irf'.

    > 'random'    : Randomize the starting values
                     using the simulated starting
                     points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double      : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing
                      with a MS-model or a break-point model you can
                      indicate the regime by 'steadystate(regime)'. E.g.
                      to start in regime 1 give; 'steadystate(1)'. For
                      MS - models the default is the ergodic mean
                      (default as long as 'girf' is not selected as
                      'type'), while for break-point models it is to
                      start from the first regime.

    > 'zero'       : Start from zero on all variables.

    > 'random'    : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'       : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'         : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'  : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'    : The variables to create impulse responses of.
                     Default is the dependent variables defined by the

```

first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_fm.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency ↑**

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB ↑**

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

ret = isStateSpaceModel(obj)

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

[bands,plotter] = jointPredictionBands(obj,varargin)

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

```

- 'vars' : A cellstr with the variables to return
the calculated joint prediction bands of.

- 'perc' : Error band percentiles. As a 1 x numErrorBand
double. E.g. [0.3,0.5,0.7,0.9] (default) or
0.9. Cannot be empty!

- 'date' : If recursive forecast has been produced,
you can use this option to choose which of
the recursive forecast to construct the
joint prediction bands of. Default is empty,
i.e. use the last forecast.

- 'method' : Choose between:

    > 'kolsrud' : See Akram et al (2016), Joint
                   prediction bands for
                   macroeconomic risk management

    > 'copula' : Using a copula likelihood approach.

    > 'mostlik' : Group the paths based on how
                   likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the
error bands. If empty (default) all forecasting steps stored
in the object is used.

```

Output:

```

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_fm.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

f = mcfRestriction(obj,type,restriction)

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

> 'irf' : A N x 4 cell, where the elements of each row are:
1. Name of the shock. 'E_X'
2. Name of the variable. 'X'
3. Horizon. E.g. 1 or 1:3.
4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.

> 'rirf' : A N x 7 cell, where the elements of each row are:
1. Name of the shock 1. 'E_X'
2. Name of the variable 1. 'X'
3. Horizon of variable 1. E.g. 1.
4. Name of the shock 2. 'E_Y'
5. Name of the variable 2. 'Y'
6. Horizon of variable 2. E.g. 2.

7. Restriction. E.g. $\@(\mathbf{x}, \mathbf{y})\gt(\mathbf{x}, \mathbf{y})$,
 $\@(\mathbf{x}, \mathbf{y})\gt(\mathbf{x}-\mathbf{y}, 0) \mid \lt(\mathbf{x}-\mathbf{y}, 1)$, i.e. a
function_handle that takes a two scalar
double as inputs and a scalar logical
as output.

> 'corr' : A N x 4 cell, where the elements of each
row are:
1. Name of the first variable.
2. Name of the second variable.
3. Number of periods to lag the 2. variable.
4. Same as 4 for 'irf'.

E.g. {'Var1', 'Var1', 1, @(\mathbf{x})\gt(\mathbf{x}, 0.1)}, to
test a restriction on the autocorrelation
at lag for variable 'Var1'.

E.g. {'Var1', 'Var2', 0, @(\mathbf{x})\lt(\mathbf{x}, 0.1)}, to
test a restriction on the contemporaneous
correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1', 'Var1', 0, @(\mathbf{x})\lt(\mathbf{x}, 0.1)}, to
test a restriction on the contemporaneous
variance.

> 'ss' : A N x 2 cell, where the elements of each
row are:
1. The expression using any of the
endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_fm.monteCarloFiltering](#), [nb_fm.irf](#)
[nb_fm.theoreticalMoments](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fm.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj, varargin)
[paramD, values, solvingFailed, objAcc] = monteCarloFiltering(obj, varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_fm.mcfRestriction](#), [nb_fm.solve](#)
[nb_fm.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterDraws ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated

with classical methods.

```

> 'wildBlockBootstrap' : Create artificial data by wild
  overlapping random block length
  bootstrap., and then "draw" parameters
  based on estimation on these data.
  Only an option for models estimated
  with classical methods.

> 'copulaBootstrap'      : Uses a copula approach to draw residual
  that are autocorrelated, Does not handle
  heteroscedasticity. Only an option for
  models estimated with classical methods.

> 'posterior'           : Posterior draws. Only for models
  estimated with bayesian methods.
  Default for models estimated with
  bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

```

- 'parallel'      : Run in parallel. true or false.

- 'cores'         : Number of cores to use. Default is to use all cores
  available.

- 'newDraws'       : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the draws input.

  Caution: When setting 'parallel' to true on MATLAB
  version later than R2017B this number will
  be the number of new draws, and not the
  factor!!! E.g. set it to 100. If it is given
  as a number less than 1, it will by default
  be set to 100.

- 'initialDraws'  : When drawing parameters this factor decides how many
  many draws that are produced before solving the model.
  Default is 1. I.e. it is equal to draws input. For
  VAR models with underidentification it may be a good
  idea to set this to a value > 1 as you may drop a
  lot of parameters when identification fails. Default
  is 1.

```

Output:

```

- out          : The output depend on the input output:

> 'param'     : Default. A struct with the following fields:

  beta   : A nPar x nEq x draws double with the

```

estimated parameters.

sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

ci = parameterIntervals(obj,alpha)

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

```
- ci      : A nPar x 3 cell matrix.  
Written by Kenneth SÃ¶terhagen Paulsen  
Inherited from superclass NB_MODEL_GENERIC
```

► **plotFactors** ↑

```
plotter = plotFactors(obj,errBands)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

```
- obj      : An object of class nb_factor_model_generic  
- errBands : Give true to add one std bands around the factor estimates.  
             Default is false.
```

Output:

```
- plotter : A nb_graph_ts object to use to plot the factors. Use the  
           graphSubPlots method or the nb_graphSubPlotGUI.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **plotForecast** ↑

```
plotter          = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects

Input:

```
- obj  : A vector of nb_model_forecast objects  
- type : Type of plot, as a string;  
         > 'default'   : Plot the forcast for the last period of  
                         estimation and ahead. Possibly with density for
```

the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.

> 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.

- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF ↑**

plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...
upperBound,method)

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fm.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fm.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fm.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosterioriors ↑**

```
plotter = plotPosterioriors(obj)
plotter = plotPosterioriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.
- Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_fm.getPosteriorDistributions](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results** ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_fm.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations** ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► set ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.

```

- 'endDate'      : End date of the decomposition. As a string. If
                  different models has different frequency this date will
                  be converted.

- 'perc'         : Error band percentiles. As a 1 x numErrorBand double.
                  E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error
                  bands.

- 'method'       : The selected method to create error bands.
                  Default is ''. See help on the method input of the
                  nb_model_generic.parameterDraws method for more
                  this input. An extra option is:

> 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent variables.
                  See nb_var.set_identification. Of course this option
                  only makes sense for VARs identified with sign
                  restrictions.

- 'replic'        : The number of simulation for posterior, bootstrap and
                  MC methods. Default is 1. Only used when 'perc' is
                  provided.

Caution: Will not have anything to say for the method
          'identDraws'. See the option 'draws' of the
          method nb_var.set_identification for more.

- 'stabilityTest' : Give true if you want to discard the draws
                  that give rise to an unstable model. Default
                  is false.

- 'parallel'      : Give true if you want to do the shock decomp in
                  parallel. Default is false.

- 'fcstDB'        : An object of class nb_ts with the forecast to
                  decompose. must contain all model variables and
                  shocks/residuals, and must start the period after
                  the estimation/filtering end date or at the
                  estimation/filtering start date. See
                  the nb_model_generic.getForecast method.

- 'type'          : Choose 'updated' or 'smoothed'. 'smoothed' is
                  default.

- 'anticipationStartDate' : Start date of anticipating shocks. As a
                  string. If different models has different
                  frequency this date will be converted.

- 'model2'         : A struct with the solution of the second
                  model to use for decomposition.

- 'secondModelStartDate' : Start date of second model. As a string.
                  If different models has different frequency
                  this date will be converted.

```

Output:

- decomp : A structure of nb_ts objects with the shock

decomposition for each model.

- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_fm.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.

```

> 'all'      : All variables are returned (but not
               the lags). Also including exogenous
               and shocks.

- 'parallel'   : Give true if you want to do the simulations in
                  parallel. I.e. spread models to different threads.
                  Default is false.

- 'parameterDraws' : Number of draws of the parameters. When set to 1
                     it will discard parameter uncertainty. Default is 1.

- 'regime'      : Select the regime you want to simulate. If empty
                  the simulation will switch between regimes. Only
                  an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations.
                  This will have no effect if the states input is
                  providing the full path of regimes. Default is 1.

- 'seed'         : Set simulation seed. Default is 2.0719e+05.

- 'startDate'    : Give the start date of the simulation. If not
                  provided the output will be a nb_data object
                  without a spesific start date (default). If
                  provided the output will be an object of class
                  nb_ts.

Caution : If dealing with break-points or exogenous
           time-varying parameters this input must be
           provided.

- 'startingValues' : Either a double or a char:

    > double       : A 1 x nVar double.

    > 'mean'        : Start from the mean of the data
                      observations. Default for all
                      model except nb_dsgc models.

    > 'steadystate' : Start from the steady state. For
                      now only an option for nb_dsgc
                      models. If you are dealing with a
                      MS-model you can indicate the
                      state by 'steadystate(state)'.
                      E.g. to start in state 1 give;
                      'steadystate(1)'. Default for
                      nb_dsgc models.

    > 'zero'         : Start from zero on all variables
                      (Except for the deterministic
                      exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                    that give rise to an unstable model. Default
                    is false.

- 'startingProb'  : Either a double or a char:

```

> double	: A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
> 'ergodic'	: Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```

[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)

```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
- Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
 - 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
 - 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is

true.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargin{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fm.graphCorrelation](#), [nb_fm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_fm object(s).

Input:

- obj : A vector of nb_fm objects.

Output:

- obj : A vector of nb_fm objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_fm.solveNormal(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_fm.solveRecursive(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_fm.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_fm.template()
options = nb_fm.template(num)
```

Description:

Construct a struct which can be provided to the nb_fm class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```
- obj : A scalar solved nb_model_generic object.  
  
- expression : A MATLAB expression as a string. Use parameter names as  
variables. See model.parameters.name.  
  
- method : A string with the method to use. For bootstrap method see  
nb_bootstrap. For bayesian models 'posterior' is the only  
option. Default is 'bootstrap' for classical models, while  
'posterior' is the default for bayesian models.  
  
- draws : Number of draws from the parameter distribution. Default  
is 1000.  
  
- alpha : Confidence level. Default is 0.05.  
  
- type : Type of test:  
    > '=' : Two sided test. expression == 0 (default)  
            For two-sided tests it is assumed that the  
            distribution is symmetric! Use  
            confidence/probability intervals instead.  
    > '>' : One sided test. expression > 0  
    > '<' : One sided test. expression < 0
```

Output:

```
- pval : > 'classical' : P-value of test.  
        > 'bayesian' : Probability of test.  
  
        A 1x1 double.  
  
- ci : A 1 x 2 double with the lower and upper bound of the  
confidence/probability interval of the tested expression.  
  
- dist : A nb_distribution object storing the distribution of the  
tested expression.
```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the

parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargout{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fm.graphCorrelation](#), [nb_fm.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.

- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.

```
> {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object  
storing the numerically calculated  
forecast error variances/skewnesses.  
  
- plotter : A nb_graph_ts object which the method graph can be used to  
produce graphs.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_fm.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_fm.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update ↑**

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.

E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_fm.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_fmdyn

Go to: [Properties](#) | [Methods](#)

A class for estimation of dynamic factor models on the form:

$$\begin{aligned} X(t) &= B \cdot W(t) + L \cdot F(t) + \epsilon(t) & (1) \\ \epsilon(t) &= \alpha \cdot \epsilon(t-1) + e(t) & (2) \\ F(t) &= A \cdot F(t-1) + u(t) & (3) \end{aligned}$$

where

- > The data is assumed to have T observations. Some may be missing, and have different frequencies.
- > $X(t)$ is a vector of observed data with size $N \times 1$.
- > $W(t)$ are the exogenous variables of the model, here the constant term will be included. $W(t)$ has size $NE \times 1$.
- > B is a matrix with size $N \times NE$.
- > $F(t)$ is the unobserved factors with size $R \cdot (nLags-1) \times 1$ (see comment on the A matrix), i.e. R is the number of factors to be estimated.
- > L is the factor loadings with size $N \times R \cdot (nLags-1)$. Zero restrictions as in Banbura et al. (2010b) may be applied, and zeros will be added to be consistent with the comment for the matrix A . L may also be adjusted to allow for mixed frequencies, again following the procedure of Banbura et al. (2010b).
- > $e(t) \sim N(0, \Sigma)$ for a diagonal covariance matrix Σ .
- > α is a diagonal matrix of AR(1) parameters of the idiosyncartic component of each variable in X . Optional. If turned off, it is assumed that $\epsilon(t) \sim N(0, \Sigma)$.
- > $u(t) \sim N(0, Q)$, and is the residuals of the VAR model of the factors.
- > A is a matrix with size $R \cdot (nLags-1) \times R \cdot (nLags-1)$, where $nLags$ is the selected number of lags of the VAR of the factors, i.e.

```

F(t) = [f(t), f(t-1), ..., f(t-nLags)].
> Let beta be the full set of parameters to estimate, and zeta(t) the
full set of unobserved state variables the model can be put on the
form

```

$$X(t) = \mu + Z(\beta) * zeta(t) \quad (4)$$

$$zeta(t) = T(\beta) * zeta(t-1) + eta(t) \quad (5)$$

The model can either be estimated using a two step frequentistic algorithm, a bayesian gibbs sampling algorithm or the algorithm suggested by Koop and Korobilis (2014) extended by Schroder and Eraslan (2021) to handle mixed frequency.

1. Follows the two step procedure of Banbura et al. (2010a). First some initial values of the factors has to be estimated by principal components on a balanced datasets of X. Here we use a spline method to fill in the missing values. As this is only used to initialize the algorithm it is not of any more importance than the speed up of the convergence of the estimation algorithm. $\epsilon(t)$ is then estimated from using (1). From there parameters are then initialized with the ols estimator based on the initial values of the factors. The first step of ML algorithm is then to filter/smooth out the $zeta(t)$ given β , and then use the expected maximum likelihood estimator on β given the filtered values of $zeta(t)$. This is done until convergance of the log likelihood. This is the algorithm used when 'estim_method' option is set to 'dfmeml'.
2. In the bayesian setting we have a prior on β . The conditional posterior of $p(zeta|X, \beta)$ and $p(\beta|X, zeta)$ are then known for priors we offer. Here we drop (t) script to mean the full series. As we have the conditional posteriors we can draw from them using the Gibbs sampler as suggested by Carter and Kohn (1994). In practice we can draw from $p(zeta|X, \beta)$ using the kalman smoother given the last draw of β . When doing gibbs sampling you may need to use a burn in face and use thinning to prevent autocorrelated draws from the posterior. For more information on the prior spesification see nb_fmdyn.priorTemplate and nb_fmdyn.setPrior. Bayesian estimation is triggered by calling nb_fmdyn.setPrior, or else the algorithm under 1 is ran. This is the algorithm used when 'estim_method' option is set to 'bdfm'. (Not yet finished!)
3. Koop and Korobilis (2014) extended by Schroder and Eraslan (2021) to handle mixed frequency. See the nb_tvpmsvEstimator package. (Not part of the open source version yet!)

Caution: Some small alteration to the referenced algorithm may have been done in special cases.

References:

Banbura and Modugno (2010a), "Maximum likelihood estimation of factor models on data sets with arbitrary pattern of missing data".

Banbura et al. (2010b), "Nowcasting"

Carter and Kohn (1994), "On Gibbs sampling for state space models", Biometrika, Volume 81, Issue 3, September 1994, Pages 541-553

Superclasses:

`nb_factor_model_generic, nb_model_generic`

Constructor:

`obj = nb_fmdyn(varargin)`

Optional input:

- See the `set` method for more on the inputs.
`(nb_model_estimate.set)`

Output:

- `obj` : An `nb_fmdyn` object.

See also:

`nb_model_generic, nb_model_estimate.set`
`nb_dfmemlEstimator.estimate, nb_bdfmEstimator.estimate`

Written by Kenneth Sæterhagen Paulsen

Properties:

- `addAutoName`
- `dependent`
- `factors`
- `options`
- `results`
- `addAutoNameIfLocal`
- `endogenous`
- `forecastOutput`
- `parameters`
- `solution`
- `addID`
- `estOptions`
- `name`
- `reporting`
- `transformations`
- `addIDIfLocal`
- `exogenous`
- `observables`
- `residuals`
- `userData`

- **`addAutoName`** ↑

Add automatic name (first) to default name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addAutoNameIfLocal`** ↑

Add automatic name (first) to local name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addID`** ↑

Add identifier to default name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addIDIfLocal`** ↑

Add identifier to local name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **factors** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the factors, the tex_name holds the names in the tex format. The number field holds a double with the number of factors. Cannot be set. Model must be estimated

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **observables** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observables',{'Var1','Var2'}); or use the <className>.template() method.

If the model class also supports mixed frequency data, it has two additional fields; frequency and mapping. These fields stores the selected frequency and mapping for each element.

Inherited from superclass NB_FACTOR_MODEL_GENERIC

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.

- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculate
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- detNumFactors
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFcstAtDates
- evaluatePrior
- explained
- forecast
- forecastPerc2ParamDist
- getCalculated
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getOriginalVariables
- getParameterDrawsMethods
- assignParameters
- assignTexNames
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- explainFactors
- factorDecomposition
- forecastPerc2Dist
- getActual
- getCondDB
- getDependent
- getEstimationOptions
- getFactors
- getForecast
- getHistory
- getModelNames
- getObservables
- getPIT
- getParameters

- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- getScore
- getVariablesList
- handleCondInfo
- help
- interpretForecast
- isDensityForecast
- isMS
- isNB
- isStatic
- isestimated
- issolved
- loadPosterior
- mincerZarnowitzTest
- parameterDraws
- plotFactors
- plotForecastDensity
- plotMCFDistTest
- plotPosteriors
- print
- print _ estimation _ results
- removeObservations
- setBlocks
- setMapping
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots
- getSolution
- graphCorrelation
- handleMissing
- initialize
- irf
- isFiltered
- isMixedFrequency
- isStateSpaceModel
- isbayesian
- isforecasted
- jointPredictionBands
- mcfRestriction
- monteCarloFiltering
- parameterIntervals
- plotForecast
- plotMCF
- plotMCFValues
- plotPriors
- printCov
- priorTemplate
- set
- setFrequency
- setPrior

- shock_decomposition
 - simulate
 - simulateFromDensity
 - simulatedMoments
 - solve
 - solveNormal
 - solveRecursive
 - solveVector
 - struct
 - template
 - testForecastRestrictions
 - testParameters
 - theoreticalMoments
 - trendAndCycle
 - uncertaintyDecomposition
 - uncondForecast
 - unstruct
 - update
 - variance_decomposition
-

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters** ↑

obj = assignParameters(obj,varargin)

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

```
obj = assignPosteriorDraws(obj,varargin)
```

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames** ↑

```
obj = assignTexNames(obj, names, texNames)
```

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as `_t` is automatically appended to each variable! Use superscripts instead.

Input:

- `obj` : An object of class `nb_model_generic`.
- `names` : A `Nx1` cellstr with the names of the variables and/or parameters of the model.
- `texNames` : A `Nx1` cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- `obj` : A `nb_model_generic` object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its `tex_name` fields. (Also `observables` and `observablesFast` for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **calculate** ↑

```
calc = calculate(obj, varargin)
```

Description:

Do the calculation associated with the object.

Input:

- `obj` : An object array of class `nb_calculate_generic`.
- `'parallel'` : Use this string as one of the optional inputs to run the estimation in parallel.

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- calc : An cell array of class nb_ts, each element stores the calculation results of the corresponding model. An element is empty if the corresponding model is found not to be valid.

See also:

[nb_fmdyn.getCalculated](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsgen.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!
- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_fmdyn.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Å;terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

obj = calculateStandardError(obj,method,draws)

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see

`nb_bootstrap`. For bayesian models '`posterior`' is the only option. Default is '`bootstrap`' for classical models, while '`posterior`' is the default for bayesian models.

- `draws` : Number of draws from the parameter distribution. Default is 1000.
- `alpha` : Confidence level. Default is 0.05.

Output:

- `obj` : A scalar `nb_model_generic` object, where the `results` property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

 - `stdBeta` : Standard deviation of coefficient of the main equation.
 - `tStatBeta` : T-statistic for the coefficient of the main equation.
 - `pValBeta` : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
 - `stdLambda` : Standard deviation of coefficient of the observation equation.
 - `tStatLambda` : T-statistic for the coefficient of the observation equation.
 - `pValLambda` : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- `ci` : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in `beta!` As a $(nCoeff * nEq) + 1 \times 3$ cell array.
- `dist` : Estimated distribution of the coefficients of the main equation. As a $(nCoeff * nEq) \times 1$ `nb_distribution` object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkModel ↑**

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosteriors ↑**

```
[DB,plotter,pAutocorr] = checkPosteriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

`nb_graph_data, nb_graphSubPlotGUI, nb_model_sampling.checkUpdatedPriors`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

`obj = checkReporting(obj)`

Description:

Check reporting, and store historical observation of reported variables.

Input:

- `obj` : A NxM `nb_modelData` object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODELDATA`

► **conditionalTheoreticalMoments** ↑

`c = conditionalTheoreticalMoments(obj,varargin)`

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the `theoreticalMoments` method
when the 'stacked' input is set to true.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Input:

- `obj` : An object of class `nb_model_generic`

Optional inputs;

- `'vars'` : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws. Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with

bayesian methods.

Caution : Posterior draws is
already made at the
estimation stage.

- 'nSteps' : Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB           = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous

restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
          allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.

- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **convert ↑**

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handled correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the

input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

```
> second column : Any expression that can be interpreted  
by the nb_ts.createVariable method.  
  
> third column : Any expression that can be interpreted  
by the nb_ts.createShift method.  
  
> fourth column : Comments  
  
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is  
important that this option is higher than the number of  
forecasting/irf steps, or else the output will be nan!
```

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name, input, shift, description  
'VAR1_G', 'pcn(VAR1)', 'avg', 'VAR1 growth'  
'VAR2_G', 'pcn(VAR2)', 'avg', 'VAR2 growth'  
'VAR3_G', 'pcn(VAR3)', 'avg', 'VAR3 growth'};  
  
model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **detNumFactors ↑**

R = detNumFactors(obj,type)

Description:

Determine the number of factors to use in the factor model. These test are based on PCA on the observed data. If missing observations these are filled in for using nb_dfmmlEstimator.fillInForMissing.

Input:

- type : One of:
 - > 'scree' : Scree plot.
 - > 'expl' : Criteria using that the factors should explain X amount of the variation in the data.
 - > 'trapani' : Uses the test that is implemented in nb_trapaniNFactorTest.
 - > 'bn' : Implements the IC_p1 test of Bai and Ng (2002).
 - > 'horn' : Implements the test proposed by Horn (1965) and developed by Ledesma et al. (2007). Code is written by Lanya Tianhao Cai (2016). This test is only valid if the time series are standardized. See the option 'transformation' of the nb_fmdyn object to apply this.

Optional input:

- 'threshold' : The threshold for the given type of test. Have an impact on the following types
 - > 'expl' : Sets the X value. Default is 0.9.
 - > 'trapani' : Confidence level of test. Default is 0.05.

Output:**See also:**

[nb_trapaniNFactorTest](#)

Written by Kenneth Å!terhagen Paulsen

► dieboldMarianoTest ↑

[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band width selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fmdyn.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist** ↑

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_fmdyn.forecast](#), [nb_fmdyn.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.

- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!

- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_fmdyn.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_fmdyn.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use

```
getVariable(varargin{i},'y','x'),  
while to get cov(x(-i),y) (== cov(x,y(+i))) you  
can use getVariable(varargin{i},'x','y'). (This  
example only works if the output is of class nb_cs)
```

See also:

[nb_fmdyn.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind     = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d,{'MSE'},0,1)
m = evalFcstAtDates(b,d,{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

```
- prior : See options.prior.  
- par : A nEst x 1 double with the value of the point of evaluation.
```

Output:

```
- logPriorD: The log prior density at the evaluated point.
```

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **explainFactors** ↑

```
[dec,plotter] = factorDecomposition(obj)
```

Description:

Decomposition of the factors into contributions from the observables.

Input:

```
- obj : A scalar nb_fmdyn object.
```

Output:

```
- explained : A nb_cs object with size nFactors x nObservables that stores the explained variance of each observable.  
- plotter : A 1 x nFactors nb_graph_cs object array. Use the graph method or the nb_graphMultiGUI class.
```

See also:

[nb_graph_cs](#), [nb_graphMultiGUI](#)

Written by Kenneth Sæterhagen Paulsen

► **explained** ↑

```
[expl,plotter] = explained(obj)
```

Description:

Get how much of the variation in the data the factors explain.

Caution: Remember that for dynamic factor models the factors are not orthogonal.

Input:

- obj : A scalar nb_factor_model_generic object.

Output:

- expl : A nb_cs object with size nFactors x 1.
- plotter : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to plot it on the screen.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **factorDecomposition ↑**

```
[dec,plotter] = factorDecomposition(obj)
```

Description:

Decomposition of the factors into contributions from the observables.

Input:

- obj : A scalar nb_fmdyn object.

Output:

- dec : A nb_ts object with size T x nObservables x nFactors that stores the decomposition.
- plotter : A nb_graph_ts object. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_graph_ts](#), [nb_graphPagesGUI](#)

Written by Kenneth Sæterhagen Paulsen

► **forecast ↑**

```

obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)

```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' : - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
Only for density forecast:
 - 'logScore' : Log score
Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.
- 'estDensity' : The method to estimate the density. As a string. Either:

```

        - 'normal' : Normal density approximation.

        - 'kernel' : Kernel density estimation. Default.

- 'varOfInterest' : A char with the variable to produce the forecast of.
Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!

- 'parallel'      : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.

- 'cores'         : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.

- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.

- 'draws'          : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.

- 'regimeDraws'    : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.

- 'newDraws'        : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.

- 'perc'            : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method'          : The selected method to create density forecast.
Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'bins'             : The length of the bins og the domain of the density.
Either;

> []      : The domain will be found. See

```

nb_getDensityPoints (default is that 1000 observations of the density is stored).

> integer : The min and max is found but the length of the bins is given by the provided integer.

> cell : Must be on the format:

```

{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...}
  
```

- lowerLimit1 : An integer, can be nan, i.e. will be found.

- upperLimit1 : An integer, can be nan, i.e. will be found.

- binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given the default domain. No warning will be given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The combineForecast method of the nb_model_group class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.

- 'endDate' : The end date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is empty this input will have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).

- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be

produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!
- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only

supported for the nb_var and nb_pitvar classes.

- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.

- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.

> 'endo' : All the endogenous variables are returned.

> 'fullendo' : All the endogenous variables are returned included the lag variables.

> 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Including exogenous and shocks.

- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.

> double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.

> 'mean' : Start from the mean of the data observations. Default.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

```

> 'zero'          : Start from zero on all variables
                    (Except for the deterministic
                     exogenous variables)

- 'startingProb' : Either a double or a char:

    > []           : If the model is filtered the
                     starting value is calculated
                     on filtered transition
                     probabilities. Otherwise the
                     ergodic mean is used.

    > double       : A nPeriods x nRegime or a 1 x
                     nRegime double. nPeriods refer to
                     the number of recursive periods to
                     forecast.

    > 'ergodic'    : Start from the ergodic transition
                     probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'states'        : Either an integer with the regime to
                     forecast/simulate in, or an object of class nb_ts
                     with the regimes to condition on. The name of the
                     variable must be 'states'.

Caution: For break-point models this is decided by
the dates of the breaks, and cannot be set
by this option!

- 'compareToRev'  : Which revision to compare to. Default is final
revision, i.e. []. Give 5 to get fifth revision.
must be an integer. Keep in mind that this number
should be larger than the nSteps input.

- 'compareTo'     : Sometime you want to evaluate the forecast of one
variable against another variable which is not
part of the model. E.g. if you use the first release
of a variable and want to compare it against the
final release. To do this you can give a cellstr
with size n x 2, where n is the number of variables
to change the the actual observations to test
against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
kernel density estimation of the density
forecast.

- 'exoProj'        : Tolerate missing exogenous variables by projecting
them by a fitted model. Only when the 'condDB'
input is empty!

Options are:

- ''   : No projection are done. Error will be

```

given if some exogenous variables are not given any conditional information.
 Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficent 0.8, while 'exoVar2' with a constant term equal to

0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

- 'seed'
: Set simulation seed. Default is `2.0719e+05`.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_fmdyn.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fmdyn.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fmdyn.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getActual](#) ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.

```
- solution : See the property solution of the nb_model_generic object.

- vars      : A cellstr with the names of the variables to get the actual
               data of.

- startFcst : The recursive forecasting periods. Either as a double
               with the indecies or a cellstr with the dates.

- nSteps    : Number of forecasting steps. If empty the sample will not
               be split.

- inputs     : A struct with some options passed to the
               nb_forecast.getActual function. Optional.

- nowcast    : Indicate how many periods there has been produced nowcast
               of.
```

Output:

```
- actual : A nSteps x nVars x nPeriods double with the actual data for
           each recursive forecast if nSteps is given, otherwise a nobs
           x nVars double.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCalculated** ↑

```
calc = getCalculated(obj)
```

Description:

Get calculated series from model.

Input:

```
- obj : A scalar nb_calculate_generic object.
```

Output:

```
- calc : An object of class nb_ts.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_CALCULATE_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provide recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_fmdyn.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by
nb_var.priorTemplate('dsge').

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

```
dependent = getDependent(obj)
```

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFactors** ↑

```
factors = getFactors(obj)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- factors : A nb_ts object with the estimated factors of the model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a $N \times 3$ cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_fmdyn.estimate](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getForecast](#) ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.
- outputType :
 - > 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.
 - The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forcast if produced. (Else it is a empty struct). Each field has the same sizes as the fcstData output.
- > 'graph' : This option can be used to make it easier to plot the recursive densities.
 - The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.
 - The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they is calculated, or else it is a empty struct.

To graph the recursive forecast use this example:

```

fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

```

- > 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simualtions. Mean will be at last page.
- For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.
- > 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.
 - If includeHist is true the actual data is added as an additional page of the nb_ts object.

```
Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).
```

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with
 size nRec x nVars. While the fcstPercData output
 will be a nb_ts object with size nRec x nVars x nSim.
 You can set the horizon to return by the optional
 input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_fmdyn.plotForecast](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj,varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

```
- obj      : A scalar nb_model_generic object.  
- varsIn : The variables that can be found in the data of the model.  
If empty, they are found at obj.options.data.variables.  
  
Caution: Only used in special cases.
```

Output:

```
- vars : A cellstr array with the variables of the model.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

```
- obj      : A N x M nb_model_forecast object.
```

Output:

```
- names : A N x M cellstr.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)  
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getObservables ↑

```
observables = getObservables(obj)
```

Description:

Get observables from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- observables : A nb_ts object with observables of the factor model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► getOriginalVariables ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODELDATA

► getPIT ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:[nb_calculatePIT](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getParameterDrawsMethods ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getParameters ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.
- Optional input
 - varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).
 - : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
 - : Give 'double' to return the value as a numerical array.
 - : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getPosteriorDistributions](#) ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.
- Optitonal input:
 - 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

```
- distr      : A 1 x numCoeff nb_distribution object.  
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.  
  
Written by Kenneth SÃ¶terhagen Paulsen  
  
Inherited from superclass NB_MODEL_GENERIC
```

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

```
- obj      : An object of class nb_model_generic (or a subclass of this class).
```

Output:

```
- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.
```

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

residual = getResidual(obj)

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

plotter = getResidualGraph(obj)

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores

- 'MLS' : Mean log score

If the object intput is a vector, the type input can also be a cellstr array wit matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursivly, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the 'varOfInterest' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the 'observables' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'. Only some specific class of models.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the acl output from SIM or THEO, the fourth input can be the acl output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_fmdyn.simulatedMoments](#), [nb_fmdyn.theoreticalMoments](#)
[nb_fmdyn.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► handleMissing ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

► help ↑

```
helpText = nb_fmdyn.help  
helpText = nb_fmdyn.help(option)  
helpText = nb_fmdyn.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores'	: The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB'	: A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model'	: A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps'	: Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer'	: The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'	: Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'	: Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'	: To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsge.
- 'periods'	: The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'	: The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'	: A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'	: If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'	: Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \logFunc$, where logFunc is either the minus

log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.

- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_fmdyn.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

[irfs,irfsBands,plotter] = irf(obj,varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against each other. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a

scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the
nb_graphSubPlotGUI class to produce the graphs in
this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.
- 'draws' : Number of draws for calculating gирf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.
- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}
- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.
- 'levelMethod' : One of the following:
 - > 'cumulative product' : cumprod(1 + X,1)
 - > 'cumulative sum' : cumsum(X,1)
 - > 'cumulative product (log)' : log(cumprod(1 + X,1))
 - > 'cumulative sum (exponential)' : exp(cumsum(X,1))

```

> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

> 'identDraws' : Uses the draws of the matrix with
  the map from structural shocks to dependent
  variables. See nb_var.set_identification. Of
  course this option only makes sense for VARs
  identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

E.g. {'Var',1,2}, {'Var',1,inf}

Caution: 2 means observation 2 of the IRF, and as
the IRF start at period 0, this means that
observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
irfs, while 'mean' normalize the mean and scale
percentiles/other draws accordingly. Default is
'draws'.

Caution: Setting this to 'mean' will not work for
reported variables that is not measured
as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
many new draws are going to be made. Default is 0.1.
I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
This option will parallelize over models. Default
is false.

- 'parallelL' : Give true if you want to do the irfs in parallel.
This option will parallelize over parameter
simulations. Only an option if numel(obj) == 1.
Default is false.

- 'pause' : true or false, Enable or disable pause option
when calculating the irfs. Default is true.

```

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!
- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.
- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char (Only for Markov-switching models):
 - > scalar : Select the the regime to take the initial transition probabilities from.

```

> double      : A draws x nRegime or a 1 x
    nRegime double. draws refer to
    the number set by the 'draws'
    input.

> 'ergodic'   : Start from the ergodic transition
    probabilities. Default for 'irf'.

> 'random'    : Randomize the starting values
    using the simulated starting
    points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

> double      : A nVar x 1 double.

> 'steadystate' : Start from the steady state. If you are dealing
    with a MS-model or a break-point model you can
    indicate the regime by 'steadystate(regime)'. E.g.
    to start in regime 1 give; 'steadystate(1)'. For
    MS - models the default is the ergodic mean
    (default as long as 'girf' is not selected as
    'type'), while for break-point models it is to
    start from the first regime.

> 'zero'       : Start from zero on all variables.

> 'random'    : Randomize the starting values using the simulated
    starting points. Default when 'type' set to
    'girf'.

- 'states'     : Either an integer with the states to produce irf in,
    or a double with the states to condition on. Must
    have size periods x 1 in the last case. Applies to
    both MS - models and break-point models.

- 'type'        : 'girf' or 'irf'. (If empty 'irf' will be used)

> 'girf'       : Generalized impulse response function
    calculated as the mean difference between
    shocking the model with a one period shock
    (1 std) and no shock at all. Use the 'draws'
    input to set the number of simulations to use
    to base the calculation on. This type of
    irfs must be used for non-linear models.

> 'irf'        : Standard irf for linear models. Calculated by
    shocking the model with a one period shock
    (1 std).

- 'variables'   : The variables to create impulse responses of.
    Default is the dependent variables defined by the
    first model.

    Caution: The variables reported by the reporting
    property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level

```

using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.'(E_X_NW')'. Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.
- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!
- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_fmdyn.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

ret = isDensityForecast(obj)

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsgc object is filtered or not.

Input:

- obj : An object of class nb_dsgc.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

```
- ret : A M x N x Q logical. An element is set to true if the model uses  
a NB toolbox solver.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

```
- obj : A scalar nb_model_generic object.
```

Output:

```
- ret : true or false. true if the estimation settings is so that the  
solution to the model is a state-space model.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

```
- obj : A scalar nb_model_estimate object.
```

Output:

```
- ret : true or false. true if the estimation settings is so that the  
model is static.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

[bands,plotter] = jointPredictionBands(obj,varargin)

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

```

- 'vars' : A cellstr with the variables to return
the calculated joint prediction bands of.

- 'perc' : Error band percentiles. As a 1 x numErrorBand
double. E.g. [0.3,0.5,0.7,0.9] (default) or
0.9. Cannot be empty!

- 'date' : If recursive forecast has been produced,
you can use this option to choose which of
the recursive forecast to construct the
joint prediction bands of. Default is empty,
i.e. use the last forecast.

- 'method' : Choose between:

    > 'kolsrud' : See Akram et al (2016), Joint
                   prediction bands for
                   macroeconomic risk management

    > 'copula' : Using a copula likelihood approach.

    > 'mostlik' : Group the paths based on how
                   likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the
error bands. If empty (default) all forecasting steps stored
in the object is used.

```

Output:

```

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_fmdyn.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

f = mcfRestriction(obj,type,restriction)

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

> 'irf' : A N x 4 cell, where the elements of each row are:
1. Name of the shock. 'E_X'
2. Name of the variable. 'X'
3. Horizon. E.g. 1 or 1:3.
4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.

> 'rirf' : A N x 7 cell, where the elements of each row are:
1. Name of the shock 1. 'E_X'
2. Name of the variable 1. 'X'
3. Horizon of variable 1. E.g. 1.
4. Name of the shock 2. 'E_Y'
5. Name of the variable 2. 'Y'
6. Horizon of variable 2. E.g. 2.

7. Restriction. E.g. `@(x,y)gt(x,y),`
`@(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a
function_handle that takes a two scalar
double as inputs and a scalar logical
as output.

> 'corr' : A N x 4 cell, where the elements of each
row are:
1. Name of the first variable.
2. Name of the second variable.
3. Number of periods to lag the 2. variable.
4. Same as 4 for 'irf'.

E.g. `{'Var1','Var1',1,@(x)gt(x,0.1)}`, to
test a restriction on the autocorrelation
at lag for variable 'Var1'.

E.g. `{'Var1','Var2',0,@(x)lt(x,0.1)}`, to
test a restriction on the contemporaneous
correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. `{'Var1','Var1',0,@(x)lt(x,0.1)}`, to
test a restriction on the contemporaneous
variance.

> 'ss' : A N x 2 cell, where the elements of each
row are:
1. The expression using any of the
endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_fmdyn.monteCarloFiltering](#), [nb_fmdyn.irf](#)
[nb_fmdyn.theoreticalMoments](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

`[test,pval,res] = mincerZarnowitzTest(obj,precision)`

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fmdyn.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering ↑**

```
paramD = monteCarloFiltering(obj, varargin)
[paramD, values, solvingFailed, objAcc] = monteCarloFiltering(obj, varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_fmdyn.mcfRestriction](#), [nb_fmdyn.solve](#)
[nb_fmdyn.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated

with classical methods.

```

> 'wildBlockBootstrap' : Create artificial data by wild
  overlapping random block length
  bootstrap., and then "draw" parameters
  based on estimation on these data.
  Only an option for models estimated
  with classical methods.

> 'copulaBootstrap'      : Uses a copula approach to draw residual
  that are autocorrelated, Does not handle
  heteroscedasticity. Only an option for
  models estimated with classical methods.

> 'posterior'           : Posterior draws. Only for models
  estimated with bayesian methods.
  Default for models estimated with
  bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

```

- 'parallel'      : Run in parallel. true or false.

- 'cores'         : Number of cores to use. Default is to use all cores
  available.

- 'newDraws'       : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the draws input.

  Caution: When setting 'parallel' to true on MATLAB
  version later than R2017B this number will
  be the number of new draws, and not the
  factor!!! E.g. set it to 100. If it is given
  as a number less than 1, it will by default
  be set to 100.

- 'initialDraws'  : When drawing parameters this factor decides how many
  many draws that are produced before solving the model.
  Default is 1. I.e. it is equal to draws input. For
  VAR models with underidentification it may be a good
  idea to set this to a value > 1 as you may drop a
  lot of parameters when identification fails. Default
  is 1.

```

Output:

```

- out          : The output depend on the input output:

> 'param'     : Default. A struct with the following fields:

  beta   : A nPar x nEq x draws double with the

```

estimated parameters.

sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

ci = parameterIntervals(obj,alpha)

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

```
- ci      : A nPar x 3 cell matrix.  
  
Written by Kenneth SÃ¶terhagen Paulsen  
  
Inherited from superclass NB_MODEL_GENERIC
```

► **plotFactors** ↑

```
plotter = plotFactors(obj,errBands)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

```
- obj      : An object of class nb_factor_model_generic  
  
- errBands : Give true to add one std bands around the factor estimates.  
             Default is false.
```

Output:

```
- plotter : A nb_graph_ts object to use to plot the factors. Use the  
           graphSubPlots method or the nb_graphSubPlotGUI.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **plotForecast** ↑

```
plotter          = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects

Input:

```
- obj  : A vector of nb_model_forecast objects  
  
- type : Type of plot, as a string;  
  
        > 'default'   : Plot the forcast for the last period of  
                      estimation and ahead. Possibly with density for
```

the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.

> 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.

- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the
figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF ↑**

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound, ...
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmdyn.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmdyn.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmdyn.monteCarloFiltering](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPosterioriors](#) ↑

```
plotter = plotPosterioriors(obj)
plotter = plotPosterioriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.
- Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_fmdyn.getPosteriorDistributions](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_fmdyn.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **priorTemplate ↑**

prior = nb_fmdyn.priorTemplate()
prior = nb_fmdyn.priorTemplate(type)
prior = nb_fmdyn.priorTemplate(type, num)

Description:

Construct a struct which can be given to the method setPrior.

The structure provided the user the possibility to set different prior options.

Input:

- type : A string;

- 'kkse' : This is the prior used in the paper by Koop and Korobilis (2014) extended by Schroder and Eraslan (2021) to handle mixed frequency.

> 'f0VarScale' : Scale factor on the variance of the prior on the initial value of factors. Default is 10. $N(0, f0VarScale \cdot I)$

> 'lambda0VarScale' : Scale factor on the variance of the prior on the initial value of the factor loadings. Default is 1. $N(0, lambdaVarScale \cdot I)$

> 'V0VarScale' : Scale factor on the mean of the prior on the initial value of the measurement equation covariance matrix. Default is 0.1. Dogmatic prior set to $V0VarScale \cdot I$.

> 'Q0VarScale' : Scale factor on the mean of the prior on the initial value of the state equation covariance matrix. Default is 0.1. Dogmatic prior set to $Q0VarScale \cdot I$.

> 'gamma' : Hyperparameter on prior variance of the coefficients of the state equations. On the form $V(i, j) = gamma ./ (ceil(j/options.nFactors).^2)$. Where V is a matrix with size option.nFactors x option.nLags*option.nFactors. Default value is 0.1.

> 'l_1m' : Starting value of: Decay factor for the measurement error variance of the monthly variables. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 0.9.

> 'l_1q' : Starting value of: Decay factor for the measurement error variance of the quarterly variables. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 0.9.

> 'l_2' : Starting value of: Decay factor for the factor error variance. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A

```

value of 1 implies constant parameters.
Default is 0.9.

> 'l_3'          : Starting value of:
Decay factor for the loadings' error
variance. A smaller value puts smaller
weight on past observations and thus
allows for faster parameter change. A
value of 1 implies constant parameters.
Default is 0.9.

> 'l_4'          : Starting value of:
Decay factor for the factor VAR
parameters' error variance.
A smaller value puts smaller
weight on past observations and thus
allows for faster parameter change. A
value of 1 implies constant parameters.
Default is 0.9.

> 'l_1_endo_update': Controls the endogenous forgetting
factors
1: l_1m and l_1q are time-varying/endogenous
0: l_1m and l_1q are constant/static
Default is 0.

> 'l_2_endo_update': Controls the endogenous forgetting
factors
1: l_2 is time-varying/endogenous
0: l_2 is constant/static
Default is 0.

> 'l_3_endo_update': Controls the endogenous forgetting
factors
1: l_3 is time-varying/endogenous
0: l_3 is constant/static
Default is 0.

> 'l_4_endo_update': Controls the endogenous forgetting
factors
1: l_4 is time-varying/endogenous
0: l_4 is constant/static
Default is 0.

- num  : Number of prior templates to make.

```

Output:

- options : A struct.

See also:

[nb_fmdyn](#)

Written by Kenneth Sæterhagen Paulsen and Maximilian Schräder

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setBlocks** ↑

obj = setBlocks(obj, blocks)

Description:

Use this function to set zero restriction in the factor loadings. This can be used to only restrict some factors to explain the observables. E.g. in this way we can have one factor that only relates to one class of observables, and in this way we can identify that factor with that specific class.

Input:

- obj : An object of class nb_fmdyn.

- blocks : A nb_cs or double with size N x R, where N is the number of observables and R is the number of factors. Give a nb_cs object to name the factors! Ordering of the nb_cs input is not important, but if given as a double the ordering is given by obj.observables.name. Set a one in the row of the observables that belong to the block of the corresponding column, otherwise 0.

Output:

- obj : An object of class nb_fmdyn, where the blocks field of the options property is set.

See also:

[nb_fmdyn.set](#)

Written by Kenneth Sæterhagen Paulsen

► **setFrequency** ↑

`obj = setFrequency(obj, freq)`

Description:

Set the frequency of the decleared observables of a formulated model.

Supported frequencies:

- 'quarterly' : 4
- 'monthly' : 12

Input:

- `obj` : An object of class `nb_fmdyn`.
- `freq` : A cellstr on the format; `{'Var1',4,'Var2',4}`. There is no need to set the frequency of those variables that has the same frequency as the data (`options.data`)!

Output:

- `obj` : An object of class `nb_fmdyn` where the frequency of a set of variables has been set. See the `observables` properties.

See also:

[nb_fmdyn.setMapping](#)

Written by Kenneth Sæterhagen Paulsen

► **setMapping** ↑

`obj = setMapping(obj, freq)`

Description:

Set the maping of the decleared `nb_fmdyn` of a formulated model. I.e. how to map the higher frequency to lower frequency in the observation equation.

See `setFrequency` for the supported frequencies.

Supported mappings (Examples are given for a monthly and quarterly mixed frequency data):

- 'levelSummed'	:	$Y(q)$	= $Y(m) + Y(m-1) + Y(m-2)$
- 'diffSummed'	:	$Y(q)$	= $Y(m) + 2*Y(m-1) + 3*Y(m-2) + 2*Y(m-3) + Y(m-4)$
- 'levelAverage'	:	$Y(q)$	= $1/3*(Y(m) + Y(m-1) + Y(m-2))$
- 'diffAverage'	:	$Y(q)$	= $1/3*Y(m) + 2/3*Y(m-1) + Y(m-2) + 2/3*Y(m-3) + 1/3*Y(m-4)$
- 'end'	:	$Y(q)$	= $Y(m)$

The default mapping that is used for all series with lower frequency is 'diffAverage'.

Input:

- obj : An object of class nb_fmdyn
- map : A cellstr on the format; {'Var1','levelSummed',...,'Var2','diffSummed'}

Output:

- obj : An object of class nb_mfvar where the mapping of a set of variables has been set. See the dependent and block_exogenous properties.

See also:

[nb_fmdyn.setFrequency](#)

Written by Kenneth Åtterhagen Paulsen

► **setPrior** ↑

obj = setPrior(obj,prior)

Description:

Set priors of a vector of nb_fmdyn objects. The prior input must either be a struct or nb_struct with same size as obj or have size 1.

The provided struct will be added to the property options as the field prior.

See [nb_fmdyn.priorTemplate](#) for the format of the struct.

Input:

- obj : A vector of nb_fmdyn objects
- prior : A struct or nb_struct with either matching numel of obj or size 1. If size is 1 all models gets the same prior settings.

Output:

- obj : A vector of nb_fmdyn objects with the priors set.

See also:

[nb_fmdyn.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will

be converted.

- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. $[0.3, 0.5, 0.7, 0.9]$. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_fmdyn.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate** ↑

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also

including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.

- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.

- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.

- 'seed' : Set simulation seed. Default is 2.0719e+05.

- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

- > double : A 1 x nVar double.
- > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
- > 'steadystate' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsge models.
- > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'startingProb' : Either a double or a char:

```

> double      : A nPeriods x nRegime or a 1 x
    nRegime double. nPeriods refer to
    the number of recursive periods to
    forecast.

> 'ergodic'   : Start from the ergodic transition
    probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```

[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)

```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
- Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
 - 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
 - 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is

true.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fmdyn.graphCorrelation](#), [nb_fmdyn.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_fmdyn object(s).

Input:

- obj : A vector of nb_fmdyn objects.

Output:

- obj : A vector of nb_fmdyn objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

tempSol = nb_fmdyn.solveNormal(results,opt)

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

tempSol = nb_fmdyn.solveRecursive(results,opt)

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

obj = solveVector(obj)

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_fmdyn.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template ↑**

```
options = nb_fmdyn.template()
options = nb_fmdyn.template(num)
```

Description:

Construct a struct which can be provided to the nb_fmdyn class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions ↑**

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```
- obj : A scalar solved nb_model_generic object.  
  
- expression : A MATLAB expression as a string. Use parameter names as  
variables. See model.parameters.name.  
  
- method : A string with the method to use. For bootstrap method see  
nb_bootstrap. For bayesian models 'posterior' is the only  
option. Default is 'bootstrap' for classical models, while  
'posterior' is the default for bayesian models.  
  
- draws : Number of draws from the parameter distribution. Default  
is 1000.  
  
- alpha : Confidence level. Default is 0.05.  
  
- type : Type of test:  
    > '=' : Two sided test. expression == 0 (default)  
            For two-sided tests it is assumed that the  
            distribution is symmetric! Use  
            confidence/probability intervals instead.  
    > '>' : One sided test. expression > 0  
    > '<' : One sided test. expression < 0
```

Output:

```
- pval : > 'classical' : P-value of test.  
        > 'bayesian' : Probability of test.  
  
        A 1x1 double.  
  
- ci : A 1 x 2 double with the lower and upper bound of the  
confidence/probability interval of the tested expression.  
  
- dist : A nb_distribution object storing the distribution of the  
tested expression.
```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the

parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fmdyn.graphCorrelation](#), [nb_fmdyn.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **trendAndCycle** ↑

[O,decomp,results] = trendAndCycle(obj,varargin)

Description:

Decompose the observed variables into deterministic, trend, cycle and noise components.

We follow Barigozzi and Luciani (2017) "Common Factors, Trends, and Cycles in Large Datasets".

The decomposition is given by:

```
O_t = Dhat_t                                (Deterministic)
      + G*PHI1*That_t                          (Trend)
      + G*PHI0*Chat_t                          (Cycle)
      + G*PHI0*(Ghat_t - H*Chat_t) + eps_t    (Noise)
```

where O_t is the observed variables, G is the factor loadings and ϵ_{st} is the idiosyncratic components found during estimation. See section 3.3 for a description of the rest of the terms in the equation.

Input:

- 'q' : The number of common shocks. $d=q-m$ will be the number of stationary common shocks.
- 'm' : The number of common trends. Same as $q-d$ in Barigozzi and Luciani (2017).
- 'nLags' : Number of lags to include in equation (20) of Barigozzi and Luciani (2017).

Output:

- O : A $T \times N \times 4$ nb_ts object. T is number of observations, N is number of observed variables. The pages represents deterministic, trend, cycle and noise components respectively.
- decomp : A struct with fields:
 - > That : A $T \times m$ nb_ts object. T is number of observations, m is the number of common trends.
 - > Chat : A $T \times (q-m)$ nb_ts object. T is number of observations, $(q-m)$ is the number of common stationary shocks (cycle).
 - > Ithat : A $T \times (r - q)$ nb_ts object. T is number of observations, m is the number of common noise.
- results : A struct with fields PHI1, PHI0 and H. See equation above.

See also:

[nb_model_generic.estimate](#)

Written by Kenneth Sæterhagen Paulsen

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output :

```
- data      : Depend on the 'method' input:  
    > 'percentiles'   : A nHor x (nExo + nRes)*2 x nVars nb_ts  
        object storing the uncertainty  
        decomposition.  
    > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object  
        storing the numerically calculated  
        forecast error variances/skewnesses.  
  
- plotter : A nb_graph_ts object which the method graph can be used to  
    produce graphs.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► uncondForecast ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

```
- obj      : A vector of nb_model_generic objects.  
- nSteps   : Number of forecasting steps. As a 1x1 double. Default is 8.
```

Optional inputs:

- See the nb_model_generic.forecast method

Output:

```
- obj      : A vector of nb_model_generic objects. See the property  
            forecastOutput.
```

See also:

[nb_fmdyn.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► unstruct ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_fmdyn.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update ↑**

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic

- type : > '' : Only update data. Default
 > 'estimate' : Update data and estimate.
 > 'solve' : Update data, estimate and solve
 > 'forecast' : Update data, estimate, solve and forecast.

- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.

- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.

- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► variance_decomposition ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.

- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_fmdyn.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_fmsa

Go to: [Properties](#) | [Methods](#)

A class for estimation and identification of step ahead factor models.

Superclasses:

[nb_factor_model_generic](#), [nb_model_generic](#)

Constructor:

obj = nb_fmsa(varargin)

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_fmsa object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Properties:

- [addAutoName](#)
- [dependent](#)
- [forecastOutput](#)
- [parameters](#)
- [solution](#)
- [addAutoNameIfLocal](#)
- [endogenous](#)
- [name](#)
- [reporting](#)
- [transformations](#)
- [addID](#)
- [estOptions](#)
- [observables](#)
- [residuals](#)
- [userData](#)
- [addIDIfLocal](#)
- [exogenous](#)
- [options](#)
- [results](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **[endogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous

variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgen models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **observables** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the variable(s) of the model observation equation, the tex_name holds the names in the tex format. The number field holds a double with the number of variables of the model observation equation. To set it use the set function. E.g. obj = set(obj,'observables',{'Var1','Var2'}); or use the <className>.template() method.

If the model class also supports mixed frequency data, it has two additional fields; frequency and mapping. These fields stores the selected frequency and mapping for each element.

Inherited from superclass NB_FACTOR_MODEL_GENERIC

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t$, $e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t$, $u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFcstAtDates
- evaluatePrior
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFactors
- getForecast
- getHistory
- getModelNames
- getObservables
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- explained
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual

- getResidualGraph
 - getRoots
 - getSolution
 - graphCorrelation
 - handleMissing
 - initialize
 - irf
 - isFiltered
 - isMixedFrequency
 - isStateSpaceModel
 - isbayesian
 - isforecasted
 - jointPredictionBands
 - mcfRestriction
 - monteCarloFiltering
 - parameterIntervals
 - plotForecast
 - plotMCF
 - plotMCFValues
 - plotPriors
 - printCov
 - removeObservations
 - shock_decomposition
 - simulateFromDensity
 - solve
 - solveRecursive
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
 - getResidualNames
 - getScore
 - getVariablesList
 - handleCondInfo
 - help
 - interpretForecast
 - isDensityForecast
 - isMS
 - isNB
 - isStatic
 - isestimated
 - issolved
 - loadPosterior
 - mincerZarnowitzTest
 - parameterDraws
 - plotFactors
 - plotForecastDensity
 - plotMCFDistTest
 - plotPosteriors
 - print
 - print_estimation_results
 - set
 - simulate
 - simulatedMoments
 - solveNormal
 - solveVector
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
-

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters** ↑

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_fmsa.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convert`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convert`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

`obj = convert(obj,freq,methods,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fmsa.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_fmsa.forecast](#), [nb_fmsa.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_fmsa.forecast, nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast, nb_fmsa.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_fmsa.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast ↑**

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **explained** ↑

```
[expl,plotter] = explained(obj)
```

Description:

Get how much of the variation in the data the factors explain.

Caution: Remember that for dynamic factor models the factors are not orthogonal.

Input:

- obj : A scalar nb_factor_model_generic object.

Output:

- expl : A nb_cs object with size nFactors x 1.
- plotter : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to plot it on the screen.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
 - 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
 - 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
 - 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
- Only for density forecast:
- 'logScore' : Log score
- Can also be a cellstr with the above listed evaluation types.
- Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.
- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density

forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.

- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length of the bins is given by the provided integer.
 - > cell : Must be on the format:
`{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...}`
 - lowerLimit1 : An integer, can be nan, i.e. will be found.
 - upperLimit1 : An integer, can be nan, i.e. will be found.
 - binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given

the default domain. No warning will be given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The `combineForecast` method of the `nb_model_group` class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- `'startDate'`
 - : The start date of forecast. Must be a string or an object of class `nb_date`. If empty the `estim_end_date + 1` will be used.
 - Caution:** If '`fcstEval`' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.
- `'endDate'`
 - : The end date of forecast. Must be a string or an object of class `nb_date`. If empty the `estim_end_date + 1` will be used.
 - Caution:** If '`fcstEval`' is empty this input will have nothing to say.
- `'saveToFile'`
 - : Logical. Save densities and domains to files. One file for each model. Default is `false` (do not).
- `'observables'`
 - : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- `'condDB'`
 - : One of the following:
 - > A `nb_ts` object with size `nSteps x nCondVar` with the information to condition on.
If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A `nb_data` object with size `nSteps x nCondVar x nPeriods` with the information to condition on. The `dataNames` property must be set to the start dates of the recursive conditional information. `nPeriods` will be the number of recursive periods.
 - > A `nb_ts` object with size `nSteps x nCondVar x 3` with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given

as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)

> A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual

to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.

> StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

> Horizon : Anticipated horizon of the shocks/residual. 1 is default.

> Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned

included the lag variables. Also including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Including exogenous and shocks.

- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.

> double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.

> 'mean' : Start from the mean of the data observations. Default.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

> 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'startingProb' : Either a double or a char:

> [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'states' : Either an integer with the regime to forecast/simulate in, or an object of class nb_ts

with the regimes to condition on. The name of the variable must be 'states'.

Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!

- 'compareToRev' : Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
- 'compareTo' : Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
 - Options are:
 - '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
 - 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.
 - Caution :** If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
 - Caution :** If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.
- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the

method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matix. For the order of the

variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_fmsa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fmsa.forecast](#)

Written by Per Bjarne Bye, Kenneth Åkerhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► forecastPerc2ParamDist ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_fmsa.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst, ...
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditinal information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_fmsa.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

```
dependent = getDependent(obj)
```

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)

Written by Kenneth SÃ¶terhagen Paulsen
```

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFactors** ↑

```
factors = getFactors(obj)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- factors : A nb_ts object with the estimated factors of the model stored.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_fmsa.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0, -1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is

returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!

Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_fmsa.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getHistory ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getLHSVars ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj,varsIn)
```

Description:

Get all left hand side variables of the model.
For factor models the observable (+ observableFast) are added as well.
The list will be sorted.
Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getObservables ↑

```
observables = getObservables(obj)
```

Description:

Get observables from a estimated nb_factor_model_generic object

Input:

- obj : An object of class nb_factor_model_generic

Output:

- observables : A nb_ts object with observables of the factor model stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► getOriginalVariables ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► getPIT ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option

is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

: Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

```
- obj : An object of class nb_model_generic.
```

Optitonal input:

```
- 'draws' : The number of draws to sample from the posterior to base  
the kernel estimation on, if empty all draws are used for  
estimation.
```

Output:

```
- distr : A 1 x numCoeff nb_distribution object.
```

```
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- residual : Either a nb_ts or a nb_cs object with predicted variable(s)  
stored.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_subplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

residual = getResidual(obj)

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

plotter = getResidualGraph(obj)

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames ↑**

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_fmsa.simulatedMoments](#), [nb_fmsa.theoreticalMoments](#)
[nb_fmsa.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_fmsa.help  
helpText = nb_fmsa.help(option)  
helpText = nb_fmsa.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.

- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_fmsa.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

[irfs, irfsBands, plotter] = irf(obj, varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects
Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the

nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or
{{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:
 - > 'cumulative product' : cumprod(1 + X,1)

```

> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
    the map from structural shocks to dependent
    variables. See nb_var.set_identification. Of
    course this option only makes sense for VARs
    identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

- 'parallelL' : Give true if you want to do the irfs in parallel.

```

This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.

- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for `nb_dsg` objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method `nb_var.set_identification` for more.

- 'settings' : Extra graphs settings given to `nb_plots` function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

```

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar      : Select the the regime to take the
                     initial transition probabilities
                     from.

    > double      : A draws x nRegime or a 1 x
                     nRegime double. draws refer to
                     the number set by the 'draws'
                     input.

    > 'ergodic'   : Start from the ergodic transition
                     probabilities. Default for 'irf'.

    > 'random'    : Randomize the starting values
                     using the simulated starting
                     points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double      : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing
                      with a MS-model or a break-point model you can
                      indicate the regime by 'steadystate(regime)'. E.g.
                      to start in regime 1 give; 'steadystate(1)'. For
                      MS - models the default is the ergodic mean
                      (default as long as 'girf' is not selected as
                      'type'), while for break-point models it is to
                      start from the first regime.

    > 'zero'       : Start from zero on all variables.

    > 'random'    : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'       : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'         : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'  : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'    : The variables to create impulse responses of.
                     Default is the dependent variables defined by the

```

first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_fmsa.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency ↑**

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB ↑**

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

ret = isStateSpaceModel(obj)

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

[bands,plotter] = jointPredictionBands(obj,varargin)

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

```

- 'vars' : A cellstr with the variables to return
the calculated joint prediction bands of.

- 'perc' : Error band percentiles. As a 1 x numErrorBand
double. E.g. [0.3,0.5,0.7,0.9] (default) or
0.9. Cannot be empty!

- 'date' : If recursive forecast has been produced,
you can use this option to choose which of
the recursive forecast to construct the
joint prediction bands of. Default is empty,
i.e. use the last forecast.

- 'method' : Choose between:

    > 'kolsrud' : See Akram et al (2016), Joint
                   prediction bands for
                   macroeconomic risk management

    > 'copula' : Using a copula likelihood approach.

    > 'mostlik' : Group the paths based on how
                   likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the
error bands. If empty (default) all forecasting steps stored
in the object is used.

```

Output:

```

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate](#).estimate, [nb_fmsa](#).assignPosteriorDraws
[nb_model_sampling](#).sample

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

f = mcfRestriction(obj,type,restriction)

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

> 'irf' : A N x 4 cell, where the elements of each row are:
1. Name of the shock. 'E_X'
2. Name of the variable. 'X'
3. Horizon. E.g. 1 or 1:3.
4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.

> 'rirf' : A N x 7 cell, where the elements of each row are:
1. Name of the shock 1. 'E_X'
2. Name of the variable 1. 'X'
3. Horizon of variable 1. E.g. 1.
4. Name of the shock 2. 'E_Y'
5. Name of the variable 2. 'Y'
6. Horizon of variable 2. E.g. 2.

7. Restriction. E.g. $\@(\mathbf{x}, \mathbf{y})gt(\mathbf{x}, \mathbf{y})$,
 $\@(\mathbf{x}, \mathbf{y})gt(\mathbf{x}-\mathbf{y}, 0) || lt(\mathbf{x}-\mathbf{y}, 1)$, i.e. a
function_handle that takes a two scalar
double as inputs and a scalar logical
as output.

> 'corr' : A N x 4 cell, where the elements of each
row are:
1. Name of the first variable.
2. Name of the second variable.
3. Number of periods to lag the 2. variable.
4. Same as 4 for 'irf'.

E.g. {'Var1', 'Var1', 1, @(\mathbf{x})gt(\mathbf{x}, 0.1)}, to
test a restriction on the autocorrelation
at lag for variable 'Var1'.

E.g. {'Var1', 'Var2', 0, @(\mathbf{x})lt(\mathbf{x}, 0.1)}, to
test a restriction on the contemporaneous
correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1', 'Var1', 0, @(\mathbf{x})lt(\mathbf{x}, 0.1)}, to
test a restriction on the contemporaneous
variance.

> 'ss' : A N x 2 cell, where the elements of each
row are:
1. The expression using any of the
endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_fmsa.monteCarloFiltering](#), [nb_fmsa.irf](#)
[nb_fmsa.theoreticalMoments](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_fmsa.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj, varargin)
[paramD, values, solvingFailed, objAcc] = monteCarloFiltering(obj, varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_fmsa.mcfRestriction](#), [nb_fmsa.solve](#)
[nb_fmsa.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterDraws ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated

with classical methods.

```

> 'wildBlockBootstrap' : Create artificial data by wild
  overlapping random block length
  bootstrap., and then "draw" parameters
  based on estimation on these data.
  Only an option for models estimated
  with classical methods.

> 'copulaBootstrap'      : Uses a copula approach to draw residual
  that are autocorrelated, Does not handle
  heteroscedasticity. Only an option for
  models estimated with classical methods.

> 'posterior'           : Posterior draws. Only for models
  estimated with bayesian methods.
  Default for models estimated with
  bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

```

- 'parallel'      : Run in parallel. true or false.

- 'cores'         : Number of cores to use. Default is to use all cores
  available.

- 'newDraws'       : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the draws input.

  Caution: When setting 'parallel' to true on MATLAB
  version later than R2017B this number will
  be the number of new draws, and not the
  factor!!! E.g. set it to 100. If it is given
  as a number less than 1, it will by default
  be set to 100.

- 'initialDraws'  : When drawing parameters this factor decides how many
  many draws that are produced before solving the model.
  Default is 1. I.e. it is equal to draws input. For
  VAR models with underidentification it may be a good
  idea to set this to a value > 1 as you may drop a
  lot of parameters when identification fails. Default
  is 1.

```

Output:

```

- out          : The output depend on the input output:

> 'param'      : Default. A struct with the following fields:

  beta   : A nPar x nEq x draws double with the

```

estimated parameters.

sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

ci = parameterIntervals(obj,alpha)

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

```
- ci      : A nPar x 3 cell matrix.  
  
Written by Kenneth SÃ¶terhagen Paulsen  
  
Inherited from superclass NB_MODEL_GENERIC
```

► **plotFactors** ↑

```
plotter = plotFactors(obj,errBands)
```

Description:

Get factors from a estimated nb_factor_model_generic object

Input:

```
- obj      : An object of class nb_factor_model_generic  
  
- errBands : Give true to add one std bands around the factor estimates.  
             Default is false.
```

Output:

```
- plotter : A nb_graph_ts object to use to plot the factors. Use the  
           graphSubPlots method or the nb_graphSubPlotGUI.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_FACTOR_MODEL_GENERIC

► **plotForecast** ↑

```
plotter          = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects

Input:

```
- obj  : A vector of nb_model_forecast objects  
  
- type : Type of plot, as a string;  
  
        > 'default'   : Plot the forcast for the last period of  
                      estimation and ahead. Possibly with density for
```

the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.

> 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.

- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the
figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF ↑**

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound, ...
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmsa.monteCarloFiltering](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmsa.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_fmsa.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPosterioriors](#) ↑

```
plotter = plotPosterioriors(obj)
plotter = plotPosterioriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.
- Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_fmsa.getPosteriorDistributions](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results** ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_fmsa.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations** ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► set ↑

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.

```

- 'endDate'      : End date of the decomposition. As a string. If
                  different models has different frequency this date will
                  be converted.

- 'perc'         : Error band percentiles. As a 1 x numErrorBand double.
                  E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error
                  bands.

- 'method'       : The selected method to create error bands.
                  Default is ''. See help on the method input of the
                  nb_model_generic.parameterDraws method for more
                  this input. An extra option is:

> 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent variables.
                  See nb_var.set_identification. Of course this option
                  only makes sense for VARs identified with sign
                  restrictions.

- 'replic'        : The number of simulation for posterior, bootstrap and
                  MC methods. Default is 1. Only used when 'perc' is
                  provided.

Caution: Will not have anything to say for the method
          'identDraws'. See the option 'draws' of the
          method nb_var.set_identification for more.

- 'stabilityTest' : Give true if you want to discard the draws
                  that give rise to an unstable model. Default
                  is false.

- 'parallel'      : Give true if you want to do the shock decomp in
                  parallel. Default is false.

- 'fcstDB'        : An object of class nb_ts with the forecast to
                  decompose. must contain all model variables and
                  shocks/residuals, and must start the period after
                  the estimation/filtering end date or at the
                  estimation/filtering start date. See
                  the nb_model_generic.getForecast method.

- 'type'          : Choose 'updated' or 'smoothed'. 'smoothed' is
                  default.

- 'anticipationStartDate' : Start date of anticipating shocks. As a
                  string. If different models has different
                  frequency this date will be converted.

- 'model2'         : A struct with the solution of the second
                  model to use for decomposition.

- 'secondModelStartDate' : Start date of second model. As a string.
                  If different models has different frequency
                  this date will be converted.

```

Output:

- decomp : A structure of nb_ts objects with the shock

decomposition for each model.

- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_fmsa.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.

```

> 'all'      : All variables are returned (but not
               the lags). Also including exogenous
               and shocks.

- 'parallel'   : Give true if you want to do the simulations in
                  parallel. I.e. spread models to different threads.
                  Default is false.

- 'parameterDraws' : Number of draws of the parameters. When set to 1
                     it will discard parameter uncertainty. Default is 1.

- 'regime'      : Select the regime you want to simulate. If empty
                  the simulation will switch between regimes. Only
                  an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations.
                  This will have no effect if the states input is
                  providing the full path of regimes. Default is 1.

- 'seed'         : Set simulation seed. Default is 2.0719e+05.

- 'startDate'    : Give the start date of the simulation. If not
                  provided the output will be a nb_data object
                  without a spesific start date (default). If
                  provided the output will be an object of class
                  nb_ts.

Caution : If dealing with break-points or exogenous
           time-varying parameters this input must be
           provided.

- 'startingValues' : Either a double or a char:

        > double       : A 1 x nVar double.

        > 'mean'       : Start from the mean of the data
                         observations. Default for all
                         model except nb_dsgc models.

        > 'steadystate' : Start from the steady state. For
                          now only an option for nb_dsgc
                          models. If you are dealing with a
                          MS-model you can indicate the
                          state by 'steadystate(state)'.
                          E.g. to start in state 1 give;
                          'steadystate(1)'. Default for
                          nb_dsgc models.

        > 'zero'        : Start from zero on all variables
                          (Except for the deterministic
                          exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                    that give rise to an unstable model. Default
                    is false.

- 'startingProb'  : Either a double or a char:

```

> double	: A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
> 'ergodic'	: Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```

[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)

```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
- Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
 - 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
 - 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is

true.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fmsa.graphCorrelation](#), [nb_fmsa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_fmsa object(s).

Input:

- obj : A vector of nb_fmsa objects.

Output:

- obj : A vector of nb_fmsa objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_fmsa.solveNormal(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_fmsa.solveRecursive(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_fmsa.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template ↑**

```
options = nb_fmsa.template()
options = nb_fmsa.template(num)
```

Description:

Construct a struct which can be provided to the nb_fmsa class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions ↑**

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```
- obj : A scalar solved nb_model_generic object.  
  
- expression : A MATLAB expression as a string. Use parameter names as  
variables. See model.parameters.name.  
  
- method : A string with the method to use. For bootstrap method see  
nb_bootstrap. For bayesian models 'posterior' is the only  
option. Default is 'bootstrap' for classical models, while  
'posterior' is the default for bayesian models.  
  
- draws : Number of draws from the parameter distribution. Default  
is 1000.  
  
- alpha : Confidence level. Default is 0.05.  
  
- type : Type of test:  
    > '=' : Two sided test. expression == 0 (default)  
            For two-sided tests it is assumed that the  
            distribution is symmetric! Use  
            confidence/probability intervals instead.  
    > '>' : One sided test. expression > 0  
    > '<' : One sided test. expression < 0
```

Output:

```
- pval : > 'classical' : P-value of test.  
        > 'bayesian' : Probability of test.  
  
        A 1x1 double.  
  
- ci : A 1 x 2 double with the lower and upper bound of the  
confidence/probability interval of the tested expression.  
  
- dist : A nb_distribution object storing the distribution of the  
tested expression.
```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the

parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_fmsa.graphCorrelation](#), [nb_fmsa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.

- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.

```
> {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object  
storing the numerically calculated  
forecast error variances/skewnesses.  
  
- plotter : A nb_graph_ts object which the method graph can be used to  
produce graphs.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_fmsa.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_fmsa.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update ↑](#)

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.

E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_fmsa.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_mfvar

Go to: [Properties](#) | [Methods](#)

A class for estimation and identification of mixed frequency VAR models.

Superclasses:

[nb_var](#), [nb_model_generic](#),

Constructor:

```
obj = nb_mfvar(varargin)
```

Optional input:

- See the set method for more on the inputs.
([nb_model_estimate.set](#))

Output:

- obj : A nb_mfvar object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_mfvar.template](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [block_exogenous](#)
- [exogenous](#)
- [options](#)
- [results](#)
- [addAutoNameIfLocal](#)
- [dependent](#)
- [factors](#)
- [parameters](#)
- [solution](#)
- [addID](#)
- [endogenous](#)
- [forecastOutput](#)
- [reporting](#)
- [transformations](#)
- [addIDIfLocal](#)
- [estOptions](#)
- [name](#)
- [residuals](#)
- [userData](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[block_exogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the block exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of block exogenous variables. To set it use the set function. E.g. obj = set(obj,'block_exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_VAR

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgen models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **factors** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the factors, the tex_name holds the names in the tex format. The number field holds a double with the number of factors. To set it use the set function. E.g. obj = set(obj,'factors',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_VAR

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t$, $e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t$, $u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- ABidentification
- appendData
- applyLongRunPriors
- assignParameters
- assignPosteriorDraws
- assignTexNames
- calculateMultipliers
- calculateStandardError
- checkModel
- checkPosteriors
- checkReporting
- conditionalTheoreticalMoments
- constructCondDB
- constructScore
- constructScoreLowFreq
- convert
- convertEach
- createVariables
- dieboldMarianoTest
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- eq
- estimate
- evalFcstAtDates
- evaluateForecast
- evaluatePrior
- filter
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getActual
- getActualLowFreq
- getCondDB
- getDSGEVARPriorMoments
- getDependent
- getDependentNames
- getEstimationOptions
- getEstimationStartDate
- getFiltered
- getForecast
- getForecastLowFreq
- getForecastVariables
- getFrequency
- getFrequencyStatic
- getHistory
- getIdentification
- getIdentifiedResidual
- getLHSVars
- getModelNames
- getModelVars
- getOriginalVariables
- getPIT

- `getParameterDrawsMethods`
- `getPosteriorDistributions`
- `getPriorDistributions`
- `getRecursiveScore`
- `getResidualGraph`
- `getRoots`
- `getScoreLowFreq`
- `getVariablesList`
- `graphCorrelation`
- `handleMissing`
- `identDraws2Solutions`
- `interpretForecast`
- `isDensityForecast`
- `isIdentified`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterDraws`
- `plotForecast`
- `plotForecastLowFreq`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `getParameters`
- `getPredicted`
- `getRecursiveEstimationGraph`
- `getResidual`
- `getResidualNames`
- `getScore`
- `getSolution`
- `grangerCausalityTest`
- `handleCondInfo`
- `help`
- `initialize`
- `irf`
- `isFiltered`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `objective`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`

- print_estimation_results
 - removeObservations
 - setFrequency
 - setMeasurementEqRestriction
 - setPrior
 - shock_decomposition
 - simulateFromDensity
 - solve
 - solveRecursive
 - stateSpace
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
 - priorTemplate
 - set
 - setMapping
 - setMixing
 - set_identification
 - simulate
 - simulatedMoments
 - solveNormal
 - solveVector
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
-

► ABidentification ↑

```
S = nb_var.ABidentification(ident,A,sigma,maxDraws,draws,ask)
```

Description:

Identification of VAR model using combination of zero and sign restrictions. See Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". Many thanks to him for supplying help!

As a extension to Binning (2013) this code also allow for magnitude restrictions.

Input:

- ident : The identification assumption, as a struct. See set_identification method of the nb_var class for more
- A : Transition matrix
- sigma : Covariance matrix of the VAR residuals.
- maxDraws : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.

- draws : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Output:

- S : A nb_struct with field W, with the stored map from structural shocks to dependent variables.

Written by Kenneth Å!terhagen Paulsen,
Subfunctions of this code are written by Andrew Binning.

Inherited from superclass NB_VAR

► **appendData ↑**

obj = appendData(obj,DB)

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODELDATA

► **applyLongRunPriors** ↑

```
obj = applyLongRunPriors(obj,H,phi)
```

Description:

Apply priors for the long run as in Giannone et. al (2014).

If the prior field of the options property is not yet assign a prior, a jeffrey prior is used as default.

Caution: Please do not call this method before nb_var.setPrior!

Input:

- obj : A N x M matrix of nb_var objects, with nDep dependent variables.

- phi : Either a nDep x 1 double vector with the prior shrinkage parameter for each equation, or a m x 2 cell. m <= nDep. The format of the cell input must be as follows;

```
{'var1',1;'var2',0.5}
```

If some dependent variables are not assign any parameter the default is 1.

- H : A q x nDep matrix, where q <= nDep. Each row is the cointegration vector to apply on the prior. If q < nDep the null space will be applied to the rest of the rows.

Or it can be a q x 1 cellstr with the cointegration relations. Again q <= nDep, and if q < nDep the null space will be applied to the rest of the rows. Please note that only dependent variables may be applied in the cointegration relations! Example:

```
{'var1 - var2','var3 + var1'}
```

Output:

- obj : A N x M matrix of nb_var objects.

See also:

[nb_mfvar.set](#), [nb_mfvar.setPrior](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_VAR

► **assignParameters** ↑

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.

- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.

- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.

- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.

- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model have been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► assignTexNames ↑

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsgen.writeTex](#), [nb_dsgen.writePDF](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► calculateMultipliers ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsgen.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_mfvar.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.

- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

obj = checkModel(obj)

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterior** ↑

[DB,plotter,pAutocorr] = checkPosterior(obj,varargin)

Description:

Plot posteriors draws in the order they were drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.

- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.

- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the

all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.

Caution : Posterior draws is already made at the estimation stage.

- 'nSteps' : Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB           = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  
horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
           allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScoreLowFreq ↑**

```
score = nb_mfvar.constructScoreLowFreq(forecastOutput,type,freq)
[score,start] = nb_mfvar.constructScoreLowFreq(forecastOutput,type,...  
    freq,highPeriod,allPeriods,startDate,endDate,...  
    rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_generic class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- freq : The frequency of the evaluated forecast. Only the variables observed at this frequency are evaluated.
- highPeriod : Give 0 to get the forecast when the data of low and high frequency is balanced, 1 if the high frequency data has one more observation, and so on. Default is 0.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.

- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_mfvar.getScoreLowFreq](#)

Written by Kenneth Sæterhagen Paulsen

► **convert ↑**

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
- > third column : Any expression that can be interpreted by the nb_ts.createShift method.
- > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
  Name,      input,    shift,    description
  'VAR1_G',  'pcn(VAR1)', 'avg',   'VAR1 growth'
  'VAR2_G',  'pcn(VAR2)', 'avg',   'VAR2 growth'
  'VAR3_G',  'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest** ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold–Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.

- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_mfvar.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj,draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_mfvar.forecast](#), [nb_mfvar.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_mfvar.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity ↑**

obj = doSimulateFromDensity(obj,draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_mfvar.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► empiricalMoments ↑

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i},'y','x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i},'x','y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_mfvar.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq ↑**

`ret = eq(obj,other)`

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;
ind     = v(1) == v
```

See also:

isequal

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► estimate ↑

```
obj      = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

nb_model_generic, solve

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d,{'MSE'},0,1)
m = evalFcstAtDates(b,d,{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsgc.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **filter** ↑

```
obj = filter(obj,varargin)
```

Description:

Filter the data with the mean parameter estimates. The default is to filter over the estimation sample, but it is possible to adjust the filtering sample by setting 'estim_start_date' and 'estim_end_date'.

Input:

- obj : An object of class nb_mfvar.

Optional input:

- 'waitbar' : Provide this one line char to make a waitbar when looping the method over different models.
- Given to the nb_model_estimate.set metod. See description on relevant options to adjust.

Output:

- obj : An object of class nb_mfvar, where the filtering results are stored in the results property.

See also:

[nb_model_generic.getFiltered](#)

Written by Kenneth Sæterhagen Paulsen

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
 - 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
 - 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
 - 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
- Only for density forecast:
- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.

- 'estDensity' : The method to estimate the density. As a string. Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
- 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density

```

forecast.

- 'method'
  : The selected method to create density forecast.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input.

- 'bins'
  : The length of the bins og the domain of the density.
  Either;

    > []      : The domain will be found. See
      nb_getDensityPoints (default is that 1000
      observations of the density is stored).

    > integer : The min and max is found but the length
      of the bins is given by the provided
      integer.

    > cell     : Must be on the format:

      {'Var1',lowerLimit1,upperLimit1,binsL1;
       'Var2',lowerLimit2,upperLimit2,binsL2;
       ...}

      - lowerLimit1 : An integer, can be nan,
                     i.e. will be found.

      - upperLimit1 : An integer, can be nan,
                     i.e. will be found.

      - binsL1      : An integer with the
                     length of the bins. Can
                     be nan. I.e. bins length
                     will be adjusted to
                     create a domain of 1000
                     elements.

Caution : The variables not included will be given
          the default domain. No warning will be
          given if a variable is provided in the
          cell but not forecast by the model. (This
          because different models can forecast
          different variables)

Caution : The combineForecast method of the
          nb_model_group class is much faster if the
          lower limit, upper limit! Or else
          it must simulate new draws and do kernel
          density estimation for each model again
          with the shared domain of all models
          densities.

- 'startDate'
  : The start date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
         the start date of the recursive forecast.
         Default is to start from the first possible

```

date.

- 'endDate' : The end date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is empty this input will have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.
If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))
If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided.

I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
 - If not provided. Model stds are used.

```

> Horizon          : Anticipated horizon of the
                     shocks/residual. 1 is default.

> Periods          : The active periods of the
                     shocks/residual. If nan, the
                     shocks/residual is active for
                     all periods. Must be given!

- 'kalmanFilter'   : Set to true to use a Kalman filter to do conditional
                     forecasting (on endogenous variables). Only
                     supported for the nb_var and nb_pitvar classes.

- 'sigma'           : A nCondVar*nSteps x nCondVar*nSteps double with the
                     correlation matrix of the endogenous variables to
                     condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix
of the multivariate distribution to draw the
endogenous variables from. As the endogenous
variables are autocorrelated all variables for all
periods must be drawn at the same time, using this
autocorrelation matrix.

- 'sigmaType'       : Type of correlation calculated. For more on this
                     input see nb_copula.type.

- 'output'          : Either 'endo' (default), 'fullendo' or 'all'. This
                     input indicates which variables to return the
                     simulation of.

    > 'endo'          : All the endogenous variables are
                     returned.

    > 'fullendo'       : All the endogenous variables are
                     returned included the lag variables.

    > 'full'            : All the variables are returned
                     included the lag variables. Also
                     including exogenous and shocks.

    > 'all'             : All variables are returned (but not
                     the lags). Including exogenous and
                     shocks.

- 'startingValues'  : Either a double or a char:

This option is mainly used for making simulations
from a model.

    > double            : A nPeriods x nVar or a 1 x nVar
                     double. nPeriods refer to the
                     number of recursive periods to
                     forecast.

    > 'mean'            : Start from the mean of the data
                     observations. Default.

    > 'steadystate'     : Start from the steady state. For

```

now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

<ul style="list-style-type: none"> - 'startingProb' : Either a double or a char: 	<ul style="list-style-type: none"> > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
	<ul style="list-style-type: none"> > [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.
	<ul style="list-style-type: none"> > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
	<ul style="list-style-type: none"> > 'ergodic' : Start from the ergodic transition probabilities.
- 'stabilityTest'	: Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'states'	: Either an integer with the regime to forecast/simulate in, or an object of class nb_ts with the regimes to condition on. The name of the variable must be 'states'.
<i>Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!</i>	
- 'compareToRev'	: Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
- 'compareTo'	: Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
 - Options are:
 - '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
 - 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR coefficients as well.
- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.
- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan) .

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a $1 \times 2*N$ cell array. N is equal to the number of exogenous variables you extrapolate. An example `{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}`. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1×1 double value or a function handle to a probability distribution to draw from.
 - 'upper' : The upper bound of the selected variable. Either a 1×1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a $nVar \times nSteps$ double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1×1 double or a $nSteps \times 1$ double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size $1 \times \text{parameterDraws}$. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_mfvar.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_mfvar.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the `nb_distribution.perc2ParamDist` function.

Input:

- `forecastOutput` : A `forecast` property of the `nb_model_forecast` class where the `forecastOutput.data` stores the forecast of the percentiles.
- `distribution` : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- `draws` : Number of draws to make from the estimated distributions.

Optional input:

- `'optimizer'` : See doc of the optimizer input to the `nb_callOptimizer` function. Default is '`fmincon`'.
- `'optimset'` : See doc of the `opt` input to the `nb_callOptimizer` function.

Output:

- `forecastOutput` : A struct on the same format as the `nb_model_forecast.forecastOutput` property

See also:

[nb_mfvar.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getActualLowFreq ↑](#)

```
actual = getActualLowFreq(obj,variable,freq,release,dates,nSteps)
```

Description:

Get the actual data of a low frequency variable to for example compare forecast to.

Input:

- obj : An object of class nb_mfvar
- variable : The variable to get the actual data on.
- freq : The low frequency to fetch.

- release : The releaseto return, empty or 0 will return the final (default).
- dates : A 1 x nDates cellstr array with the dates for where to split the history.
- nSteps : The number of steps of each split. Default is 1.

Output:

- actual : A nb_ts object with the wanted release, or if nargin > 3 a nb_data object with size nSteps x nVars x nDates.

Written by Kenneth Sæterhagen Paulsen

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provided recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default

is false.

- `recStart` : An object of class `nb_date` or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if `recursive` is set to true.

Output:

- `condDB` : As a `nb_ts` or `nb_data` object. Can be given as input to the `condDB` option of the `nb_model_generic.forecast` method.
- `obj` : An object of class `nb_model_generic`. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.`data` option may change.

See also:

[nb_modelData.createVariables](#), [nb_mfvar.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by `nb_var.priorTemplate('dsge')`.

Input:

- `obj` : An object of class `nb_model_generic`.
- `dsgeVar` : A `nb_var` object set up with options for estimation of a DSGE-VAR.

Optional input:

- `'maxIter'` : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- `'tol'` : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

```
dependent = getDependent(obj)
```

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsgen.filter](#), [nb_mfvar.estimate](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

```

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with
 size nRec x nVars. While the fcstPercData output
 will be a nb_ts object with size nRec x nVars x nSim.
 You can set the horizon to return by the optional
 input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_mfvar.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastLowFreq ↑**

```

[fcstData,fcstDates]           = getForecastLowFreq(obj,freq)
[fcstData,fcstDates,fcstPercData] = getForecastLowFreq(obj,freq,...)
                                 outputType)
[fcstData,fcstDates,fcstPercData] = getForecastLowFreq(obj,freq,...)
                                 outputType,includeHist,variables)

```

Description:

Get the forecast at a low frequency as a nb_ts object.

Caution : In the case that low frequency variables are observed with
 one or more periods of data at a given recursion, these
 variables will be given nan for the number of extra periods of
 data for the given recursion.

Input:

- obj : An object of class nb_mfvar. Must have produced forecast.
- freq : The returned frequency of the forecast. Only the variables observed at this frequency are returned.
- outputType :
 - > 'recursive' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.
- > 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
- includeHist : Give true (1) to include smoothed estimates of the history in the output. Only an option for the 'date' outputType. Default is false (0).
- variables : The variables to get the forecast of, as a cellstr. When this is given the original frequency of the variables are disregarded

Output:

- fcstData : See the input outputType
- fcstDates : The forecast dates for each variable, in the case that if the outputType is a date. Otherwise the start date of the forecast of all the variables for the given recursion.
- fcstPercData : See the input outputType

See also:

[nb_model_generic.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **getForecastVariables ↑**

`vars = getForecastVariables(obj)`

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getFrequency** ↑

```
freq = getFrequency(obj,date)
```

Description:

Get the frequency of the variables of the model.

Input:

- obj : A scalar object of class nb_mfvar.
- date : As the frequency may change for some variable, you can give a date to get the frequency at a given time. Default is to return the end frequency.

Output:

- freq : A 1 x nDep + nBlockExo double with the frequency of each dependent (nDep) and block exogenous variables (nBlockExo).
 - 1 : Yearly
 - 2 : Semi-annually
 - 4 : Quarterly
 - 12 : Monthly
 - 52 : Weekly

See also:

[nb_mfvar.setFrequency](#)

Written by Kenneth Sæterhagen Paulsen

► **getFrequencyStatic** ↑

```
[freq,date] = nb_mfvar.getFrequencyStatic(estOptions,dateInd)
```

Description:

Get the frequency of the variables of the model.

See also:

[nb_mfvar.getFrequency](#)

Written by Kenneth Sæterhagen Paulsen

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getIdentification** ↑

```
matrix = getIdentification(obj)
```

Description:

Get restrictions on the structural shock matrix, i.e C in the equation below:

$$y(t) = A*y(t-1) + B*x(t) + C*e(t), \text{ where } e(t) \sim N(0, I).$$

Input:

- obj : A identified scalar nb_var object.

Output:

- matrix : nEndo x nEndo cell array.

See also:

[nb_mfvar.set_identification](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_VAR

► [getIdentifiedResidual ↑](#)

```
residual = getIdentifiedResidual(obj)
```

Description:

Get identified residual from a estimated nb_var object

If we have a VAR it can be written as (dropping exogenous variables):

$$y_t = A*y_{t-1} + e_t$$

A identified VAR can be written as:

$$y_t = A*y_{t-1} + C*d_t$$

Therefore the identified residuals/shocks can be found as:

$$d_t = \text{inv}(C)*e_t$$

Caution: Will only return the residual that uses the full sample. I.e. when recursive estimation is done, only the residuals from the last estimation will be returned.

Input:

- obj : An object of class nb_var

Output:

```
- residual : A nb_ts object with the identified residual(s) stored.  
As a nobs x nEq x nIdent. Where nIdent is the number of  
identified C matrices, i.e. when the user have asked to  
identify more than one matrix (underidentification).
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **getLHSVars** ↑

```
vars = getLHSVars(obj)  
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgen object only the observables are returned!

Input:

```
- obj : A scalar nb_model_generic object.  
- varsIn : The variables that can be found in the data of the model.  
If empty, they are found at obj.options.data.variables.
```

Caution: Only used in special cases.

Output:

```
- vars : A cellstr array with the variables of the model.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given
the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.

- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT ↑**

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:[nb_calculatePIT](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getParameterDrawsMethods ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getParameters ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.
- Optional input
 - varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).
 - : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
 - : Give 'double' to return the value as a numerical array.
 - : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getPosteriorDistributions](#) ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.
- Optitonal input:
 - 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

```
- distr      : A 1 x numCoeff nb_distribution object.  
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.  
  
Written by Kenneth SÃ¶terhagen Paulsen  
  
Inherited from superclass NB_MODEL_GENERIC
```

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- residual : Either a nb_ts or a nb_cs object with predicted variable(s)  
           stored.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPriorDistributions** ↑

```
[distr,paramNames] = getPriorDistributions(obj)
```

Description:

Get prior distributions of B-VAR model.

Input:

```
- obj      : An object of class nb_var.
```

Output:

```
- distr      : A 1 x numCoeff nb_distribution object.  
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_VAR

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)  
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error

```

        - 'MEAN'   : Mean error
        - 'STD'    : Standard error of the forecast error
        - 'ESLS'   : Exponential of the sum of the log scores
        - 'EELS'   : Exponential of the mean of the log scores
        - 'MLS'    : Mean log score

- dim          : Either 'variables' to get the recursive scores of all
                  the variables forecast by a model (the obj input must
                  be a scalar object), or else the name of the variable
                  to get the score of (must be part of all models!).

- startDate    : The date to begin constructing the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- endDate      : The date to end constructiong the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- invert        : true | false. Default true. If true it will invert the
                  score for 'RMSE', 'MSE', 'MAE' and 'STD'.

```

Output:

```

- score     : A nb_ts object with size nPeriods x nVars x nHorizon if dim
               equal 'variables', else a nb_ts object with size nPeriods x
               nObj x nHorizon.

- plotter   : A nb_graph_ts object to plot the recursive scores. Use the
               graph method. One figure per horizon!

```

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Åstrehagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual ↑**

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph ↑**

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which can you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames ↑**

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

roots = getRoots(obj)

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```

scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  

                  rollingWindow,lambda)

```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
 - type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- If the object input is a vector, the type input can also be a cellstr array with matching length.
- Caution: See comment to the invert input!
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
 - startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
 - endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
 - invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
 - rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
 - lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getScoreLowFreq ↑**

```
scores = getScore(obj,type,freq)
scores = getScore(obj,type,freq,highPeriod,allPeriods,startDate, ...
                  endDate,invert,rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation. Nowcast is not evaluated, in this case see [nb_mfvar.getForecastLowFreq](#)!

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- freq : The frequency of the evaluated forecast. Only the variables observed at this frequency are evaluated.
- highPeriod : Give 0 to get the forecast when the data of low and high frequency is balanced, 1 if the high frequency data has one more observation, and so on. Default is 0.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date. Must be on the frequency of the most

frequent data used.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date. Must be on the frequency of the most frequent data used.
- invert : false or true. Default is false. Default is to report the score so that high value means good model.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

► **getSolution ↑**

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the 'varOfInterest' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the 'observables' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'. Only some specific class of models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **grangerCausalityTest** ↑

```
[test,pval,results] = grangerCausalityTest(obj,precision)
```

Description:

Test for granger causality between variables using a joint F-test that all the parameters of the lags of a variable y is zero in the equation where x is the dependent variable. If the hypothesis can be rejected we say that y granger cause x.

Caution: If recursive estimation is done it will anyway only do the test using the parameters estimated over the full sample.

Input:

- obj : A 1 x 1 nb_var object
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : The test statistics. As a nDep x nDep double. The diagonal will return nan.
- pval : The p-value of the test statistics. As a nDep x nDep double. The diagonal will return nan.
- results : A string with the printed results of the test.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► graphCorrelation ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_mfvar.simulatedMoments](#), [nb_mfvar.theoreticalMoments](#)

`nb_mfvar.empiricalMoments`, `nb_graphPagesGUI`, `nb_graph_cs`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **handleCondInfo** ↑

`ret = handleCondInfo(obj)`

Description:

Check if a `nb_model_estimate` object handle conditional info.

Input:

- `obj` : A scalar `nb_model_etimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

► **handleMissing** ↑

`ret = handleMissing(obj)`

Description:

Check if a `nb_model_estimate` object handle missing observations.

Input:

- `obj` : A scalar `nb_model_etimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **help** ↑

```
helpText = nb_mfvar.help  
helpText = nb_mfvar.help(option)  
helpText = nb_mfvar.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **identDraws2Solutions** ↑

```
sol = identDraws2Solutions(obj)
```

Description:

This function assumes that you have used the set_identification function with the input 'type' set to 'combination'.

Input:

- obj : An object of class nb_var.

Output:

- sol : A struct storing the solution of the model. Same format as the solution property of the nb_var class.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_VAR

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given,

otherwise this option does not apply.

- 'optimizer'
 - : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'
 - : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'
 - : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'
 - : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods'
 - : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'
 - : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'
 - : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'
 - : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'
 - : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights'
 - : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_mfvar.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgc objects.
- 'compare' : Give true if you have a scalar nb_dsgc model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

```

- 'draws' : Number of draws for calculating girf. Default is
           1000. For Markov switching models this will set the
           number of simulated paths of states. For more see
           the 'type' input.

- 'factor' : A cell array with the factors to multiply the
             irf of the individual model variables.
             I.e. {'var1',100,...} or
                  {'var1','var2'},100,...}

If the string starts with a asterisk you will
multiply all the variables that contain that
string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This option sets the error band percentiles of the
               graph, when the 'perc' input is empty. As a 1 x
               numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
               Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from
                  this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

    > 'cumulative product' : cumprod(1 + X,1)

    > 'cumulative sum' : cumsum(X,1)

    > 'cumulative product (log)' : log(cumprod(1 + X,1))

    > 'cumulative sum (exponential)' : exp(cumsum(X,1))

    > 'cumulative product (%)' : cumprod(1 + X/100,1)

    > 'cumulative sum (%)' : cumsum(X/100,1)

    > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))

    > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))

    > '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
             Default is ''. See help on the method input of the
             nb_model_generic.parameterDraws method for more
             this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
                    the map from structural shocks to dependent

```

variables. See `nb_var.set_identification`. Of course this option only makes sense for VARs identified with sign restrictions.

- `'normalize'` : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to `inf`, it will normalize the max impact period.
E.g. `{'Var',1,2}, {'Var',1,inf}`
Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- `'normalizeTo'` : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- `'newReplic'` : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- `'parallel'` : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- `'parallelL'` : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.
- `'pause'` : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- `'perc'` : Error band percentiles. As a 1 x `numErrorBand` double. The input must be given as the coverage, and not the percentiles itself. E.g. `[0.3,0.5,0.7,0.9]`. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So `[0.3,0.5,0.7,0.9]` will get the percentiles `[5,15,25,35,65,75,85,95]`, i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- `'periods'` : Number of periods of the impulse responses.
- `'plotSS'` : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.

```

- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be
a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws
that give rise to an unstable model. Default
is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):
    > scalar : Select the the regime to take the
initial transition probabilities
from.

    > double : A draws x nRegime or a 1 x
nRegime double. draws refer to
the number set by the 'draws'
input.

    > 'ergodic' : Start from the ergodic transition
probabilities. Default for 'irf'.

    > 'random' : Randomize the starting values
using the simulated starting
points. Default for 'girf'.

- 'startingValues' : Either a double or a char:
    > double : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing

```

with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

```

> 'zero'          : Start from zero on all variables.

> 'random'        : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'         : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'           : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generalized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'   : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'       : The variables to create impulse responses of.
                     Default is the dependent variables defined by the
                     first model.

Caution: The variables reported by the reporting
          property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                     using the method specified by 'levelMethod'.

```

Output:

```

- irfs            : The (central) impulse response function stored in a
                     structure of nb_ts object. Each field stores the the
                     impulse responses of each shock. Where the variables
                     of the nb_ts object is the impulse responses of the
                     wanted endogenous variables.

                     I.e. the impuls responses to the shock 'E_X_NW' is
                     stored in irfs.('E_X_NW'). Which will then be a nb_ts
                     object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
          responses from each model are saved as
          different datasets (pages) of the nb_ts object.

```

```

Caution: Each nb_model_generic object can have different
variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store
  a nb_ts object with the error bands of all variables. The
  pages of the nb_ts object is the percentiles (from lower
  to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter   : An object of class nb_graph_ts.

  > 'compareShocks' set to false (default)
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

  > 'compareShocks' set to true
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

- obj       : If the process has been paused, some temporary output will
  be stored in the object, and if you want to continue at a
  later stage you can take up from this point using this
  returned output. If not paused this output is empty.

```

See also:

[nb_mfvar.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isIdentified** ↑

```
ret = isIdentified(obj)
```

Description:

Is the nb_var object identified or not.

Input:

- obj : A nb_var object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsg objects!

For nb_dsg objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

ret = isStateSpaceModel(obj)

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.
- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.
- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

posterior = loadPosterior(obj)

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_mfvar.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► mcfRestriction ↑

f = mcfRestriction(obj,type,restriction)

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

> 'irf' : A N x 4 cell, where the elements of each row are:
1. Name of the shock. 'E_X'
2. Name of the variable. 'X'
3. Horizon. E.g. 1 or 1:3.
4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a
function_handle that takes a scalar
double as input and a scalar logical as
output.

> 'rirf' : A N x 7 cell, where the elements of each

```

row are:
1. Name of the shock 1. 'E_X'
2. Name of the variable 1. 'X'
3. Horizon of variable 1. E.g. 1.
4. Name of the shock 2. 'E_Y'
5. Name of the variable 2. 'Y'
6. Horizon of variable 2. E.g. 2.
7. Restriction. E.g. @(x,y)gt(x,y),
@(x,y)gt(x-y,0)||lt(x-y,1), i.e. a
function_handle that takes a two scalar
double as inputs and a scalar logical
as output.

> 'corr' : A N x 4 cell, where the elements of each
row are:
1. Name of the first variable.
2. Name of the second variable.
3. Number of periods to lag the 2. variable.
4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to
test a restriction on the autocorrelation
at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to
test a restriction on the contemporaneous
correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to
test a restriction on the contemporaneous
variance.

> 'ss' : A N x 2 cell, where the elements of each
row are:
1. The expression using any of the
endogenous variables of the model.
2. Same as 4 for 'irf'.

```

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_mfvar.monteCarloFiltering](#), [nb_mfvar.irf](#)
[nb_mfvar.theoreticalMoments](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest** ↑

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_mfvar.uncondForecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_mfvar.mcfRestriction](#), [nb_mfvar.solve](#)

`nb_mfvar.assignParameters`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► `objective` ↑

```
fval = nb_mfvar.objective(par,indPar,beta,modelInfo,options,y,...  
                           observables)
```

Description:

The objective to minimize when doing estimation of a mixed frequency VAR.

Input:

- `par` : Current vector of the estimated parameters.
- `indPar` : Not needed. Just for generics.
- `beta` : Not needed. Just for generics.
- `model` : Not needed. Just for generics.
- `options` : The `obj.estOptions` property of the `nb_dsg` class.
(After calling `getEstimationOptions`, i.e. calling the `estimate` method.)
- `y` : A `nObservables` x `nPeriods` double with the data to estimate the model on. The data may contain missing observations.
- `z` : A `nExo` x `nPeriods` double with the data of the exogenous variables of the model.
- `observables` : Not needed. Just for generics.

Output:

- `fval` : Value of the objective at the given parameters.

See also:

`nb_mfvar.stateSpace`, `nb_kalmanlikelihood_missing`
`nb_model_generic.estimate`

Written by Kenneth Sæterhagen Paulsen

► `parameterDraws` ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length

bootstrap., and then "draw" parameters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'copulaBootstrap' : Uses a copula approach to draw residual
 that are autocorrelated, Does not handle
 heteroscedasticity. Only an option for
 models estimated with classical methods.

> 'posterior' : Posterior draws. Only for models
 estimated with bayesian methods.
 Default for models estimated with
 bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
 on this option.
- stable : Give true if you want to discard the parameter draws
 that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

 Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
- > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

```
lambda   : Estimated coefficients of the observation
            equation. As a nFac x nObservables x draws
            double.

R        : Estimated covariance matrix of the
            observation equation. As a nFac x nFac x
            draws double.

factors  : Draws of the factors as a T x nFac x
            draws double.

> 'solution' : Get the solution of the model for each draw from the
                distribution of parameters. The output will be a struct
                with size 1 x draws. The struct will have the same
                format as the solution property of the underlying
                object. I.e. only one output!

> 'object'   : Get the draws as a 1 x draws nb_model_generic object.
                Each element will be a model representing a given draw
                from the distribution of the parameters. I.e. only one
                output!
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + ( +/- nb_distribution.t_icdf(alpha/2)) * stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotForecast** ↑

```
plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forcast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastLowFreq** ↑

```
plotter = plotForecastLowFreq(obj,freq)
[plotter,plotFunction2Use] = plotForecastLowFreq(obj,freq,type,...)
                           startDate,highPeriod)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_generic objects

Input:

- obj : A vector of nb_mfvar objects.
- freq : The frequency of the forecast to plot. Only the variables observed at this frequency are plotted.
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced. Must be at the high frequency.
- highPeriod : Give 0 to get the forecast when the data of low and high frequency is balanced, 1 if the high frequency data has one more observation, and so on. Default is 0.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.forecast](#), [nb_model_forecast.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

► [plotMCF ↑](#)

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound, ...
                                    upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.

- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_mfvar.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest ↑**

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo

filtering.

- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_mfvar.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues ↑**

plotter = nb_model_generic.plotMCFValues(paramD, params, nameValue)

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_mfvar.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optitonal input:

- `'prior'` : Give this string as an input to compare posterior draws with the prior distributions (if available).
- `'updated'` : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- `'subplot'` : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- `'draws'` : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If `'updated'` is given as input, this will also set the the same options for the updated prior!

Output:

- `plotter` : A $1 \times \text{numCoeff}$ vector of objects of class `nb_graph_data`. Use either the `graph` method or the `nb_graphMultiGUI` class to plot the posteriors on screen.

Caution: If `'subplot'` is given it will return a scalar `nb_graph_data` object. Use the `graphSubPlots` method or the `nb_graphSubPlotGUI` class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_mfvar.getPosteriorDistributions](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPriors** ↑

```
plotter = plotPriors(obj,varargin)
```

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov ↑**

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_mfvar.print](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **priorTemplate** ↑

```
prior = nb_mfvar.priorTemplate()
prior = nb_mfvar.priorTemplate(type)
prior = nb_mfvar.priorTemplate(type, num)
```

Description:

Construct a struct which can be given to the method setPrior.

The structure provided the user the possibility to set different prior options.

Input:

- type : A string;
 - 'minnesotaMF' : Minnesota type prior options.
 - > 'a_bar_1' : Hyperparameter on own lags. Default is 0.5.
 - > 'a_bar_2' : Hyperparameter on other lags. Default is 0.5.
 - > 'a_bar_3' : Hyperparameter on other lags. Default is 100.
 - > 'ARcoeff' : Hyperparameter on first lag coefficient of each equation. Default is 0.9.
 - > 'coeff' : A N x 2 cell array with specific priors on some coefficients. In the first column you must provide the name of the coefficient, name of dependent + _ + name of rhs variable + lag specifier. E.g. 'Var1_Var1_lag1' or 'Var1_Var2'. In the last example Var2 is an exogenous variable. In the second column you give the prior value, as a scalar double.
 - > 'method' : Sets the method to use to draw from the posterior. Either 'default' (default) or any other string. 'default' uses (a fixed) covariance matrix of the shocks (i.e. the prior). Otherwise it also samples from the posterior distribution of the covariance matrix, as in the same way as in the case of the independent normal wishart (inwishart) prior. The 'burn', 'thin' and 'S_scale' options only applies to this last case.

```

> 'burn'      : How many draws that should be used as burn
in. Default is 500.

> 'thin'      : Every k draws are kept when doing gibbs.
This options sets k. Default is 2. Increase
this option to prevent autocorrelated
draws. See nb_model_generic.checkPosterior
to test for autocorrelated draws.

> 'S_scale'   : Prior scale of sigma, i.e. the covariance
matrix of the residuals. Default is 1.

> 'R_scale'   : Inverse prior scale of R, i.e. the covariance
matrix of the measurement errors. Default is
10. Default is to only used it for high
frequency variables, if series are measured
with more than one frequency. To apply it to
specific variables use a cell array. In the
first column you must provide the name of
dependent variable. E.g. 'Var1'. In the
second column you give the prior value,
as a scalar double. Full example
{'Var1',inf;'Var2',10}. Here you allow for no
measurement error in 'Var1', and some
measurement error in 'Var2'. All variables
not provided when a cell is used, are set to
inf.

The posterior is constructed by dividing the
variance of each series by the scaling
parameters.

> 'mixing'    : Set to 'low' to use the low frequency
observations when forming the prior of the
covariance matrix of the residuals. Only
in use if you have declared variables as
mixing. Default is to use the high frequency
variable ('high').

- 'nwishartMF' : Normal-Wishart prior options.

> 'V_scale'   : Scale of the prior of the variance of the
coefficients. Default is 10.

> 'S_scale'   : Prior scale of sigma, i.e. the covariance
matrix of the residuals. Default is 1.

> 'burn'      : How many draws that should be used as burn
in. Default is 500.

> 'thin'      : Every k draws are kept when doing gibbs.
This options sets k. Default is 2. Increase
this option to prevent autocorrelated
draws. See nb_model_generic.checkPosterior
to test for autocorrelated draws.

> 'R_scale'   : Inverse prior scale of R, i.e. the covariance
matrix of the measurement errors. Default is
10. Default is to only used it for high

```

frequency variables, if series are measured with more than one frequency. To apply it to specific variables use a cell array. In the first column you must provide the name of dependent variable. E.g. 'Var1'. In the second column you give the prior value, as a scalar double. Full example {'Var1',inf;'Var2',10}. Here you allow for no measurement error in 'Var1', and some measurement error in 'Var2'. All variables not provided when a cell is used, are set to inf.

The posterior is constructed by dividing the variance of each series by the scaling parameters.

- 'glpMF' : This is the prior used in the paper by Giannone, Lenza and Primiceri (2014) adapted to the mixed frequency setting. Cannot apply block exogenous variables with this prior.
- > 'ARcoeff' : Hyperparameter on first lag coefficient of each equation. Default is 0.9.
- > 'coeff' : See same options for the 'minnesota' prior.
- > 'lambda' : Hyperparameter that controls the overall tightness of this prior. Default is 0.2.
- > 'Vc' : Hyperparameter on exogenous variables. Default is 1e7.
- > 'S_scale' : Prior scale of prior on sigma. Default is 1, i.e. OLS.
- > 'burn' : How many draws that should be used as burn in. Default is 500.
- > 'thin' : Every k draws are kept when doing gibbs. This option sets k. Default is 2. Increase this option to prevent autocorrelated draws. See nb_model_generic.checkPosterior to test for autocorrelated draws.
- 'inwishartMF' : Independent Normal-Wishart prior options.
 - > 'V_scale' : Scale of the prior of the variance of the coefficients. Default is 10.
 - > 'S_scale' : Prior scale of sigma, i.e. the covariance matrix of the residuals. Default is 1.
 - > 'burn' : How many draws that should be used as burn in. Default is 500.
 - > 'thin' : Every k draws are kept when doing gibbs. This option sets k. Default is 2. Increase this option to prevent autocorrelated

draws. See nb_model_generic.checkPosteriors to test for autocorrelated draws.

> 'R_scale' : Inverse prior scale of R, i.e. the covariance matrix of the measurement errors. Default is 10. Default is to only used it for high frequency variables, if series are measured with more than one frequency. To apply it to specific variables use a cell array. In the first column you must provide the name of dependent variable. E.g. 'Var1'. In the second column you give the prior value, as a scalar double. Full example {'Var1',inf;'Var2',10}. Here you allow for no measurement error in 'Var1', and some measurement error in 'Var2'. All variables not provided when a cell is used, are set to inf.

The posterior is constructed by dividing the variance of each series by the scaling parameters.

- 'kkse' : This is the prior used in the paper by Koop and Korobilis (2014) extended by Schroder and Eraslan (2021) to handle mixed frequency, but adapted to the VAR setting.
- > 'f0VarScale' : Scale factor on the variance of the prior on the initial value of factors. Default is 10. N(0,f0VarScale*I)
- > 'lambda0VarScale' : Scale factor on the variance of the prior on the initial value of the factor loadings. Default is 1. N(0,lambdaVarScale*I). !!Remove!!
- > 'V0VarScale' : Scale factor on the mean of the prior on the initial value of the measurement equation covariance matrix. Default is 0.1. Dogmatic prior set to V0VarScale*I. !!Remove!!
- > 'Q0VarScale' : Scale factor on the mean of the prior on the initial value of the state equation covariance matrix. Default is 0.1. Dogmatic prior set to Q0VarScale*I.
- > 'gamma' : Hyperparameter on prior variance of the coefficients of the state equations. On the form $V(i,j) = \text{gamma} / (\text{ceil}(j/\text{options.nFactors})^2)$. Where V is a matrix with size option.nFactors x option.nLags*option.nFactors. Default value is 0.1.
- > 'l_1m' : Decay factor for the measurement error variance of the monthly variables. A

smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 1. Do not adjust!

> 'l_1q' : Decay factor for the measurement error variance of the quarterly variables. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 1. Do not adjust!

> 'l_2' : Decay factor for the factor error variance. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 0.9.

> 'l_3' : Decay factor for the loadings' error variance. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 1. Do not adjust!

> 'l_4' : Decay factor for the factor VAR parameters' error variance. A smaller value puts smaller weight on past observations and thus allows for faster parameter change. A value of 1 implies constant parameters. Default is 0.9.

> 'l_1_endo_update': Controls the endogenous forgetting factors
 1: l_1m and l_1q are time-varying/endogenous
 0: l_1m and l_1q are constant/static
 Default is 0. Do not adjust!

> 'l_2_endo_update': Controls the endogenous forgetting factors
 1: l_2 is time-varying/endogenous
 0: l_2 is constant/static
 Default is 0.

> 'l_3_endo_update': Controls the endogenous forgetting factors
 1: l_3 is time-varying/endogenous
 0: l_3 is constant/static
 Default is 0. Do not adjust!

> 'l_4_endo_update': Controls the endogenous forgetting factors
 1: l_4 is time-varying/endogenous
 0: l_4 is constant/static
 Default is 0.

- num : Number of prior templates to make.

Output:

- options : A struct.

See also:

[nb_mfvar](#), [nb_model_generic.checkPosteriors](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations** ↑

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setFrequency** ↑

obj = setFrequency(obj, freq)

Description:

Set the frequency of the decleared dependent and/or block_exogenous variables of a formulated model.

Supported frequencies:

- 'yearly' : 1
- 'quarterly' : 4
- 'monthly' : 12
- 'weekly' : 52

Input:

- obj : An object of class nb_mfvar.
- freq : A cellstr on the format; {'Var1',1,'Var2',4}. There is no need to set the frequency of those variables that has the same frequency as the data (options.data)!

You can also declare that one variable change frequency using the syntax {'Var1',1,'Var2',{4,cDate,12}}. Here date must be a nb_date object or date string on the same frequency as the data property. It is interpreted as the variable 'Var2' has quarterly frequency until and including the date cDate, from there it has monthly frequency. See example NBTOOLBOX\... Examples\Econometrics\MixedFrequency\test_nb_mfvar_changeFreq.m

Output:

- obj : An object of class nb_mfvar where the frequency of a set of variables has been set. See the dependent and block_exogenous properties.

See also:

[nb_mfvar.setMapping](#)

Written by Kenneth Sæterhagen Paulsen

► [setMapping ↑](#)

obj = setMapping(obj,map)

Description:

Set the maping of the decleared dependent and/or block_exogenous variables of a formulated model. I.e. how to map the higher frequency to lower frequency in the observation equation.

See [setFrequency](#) for the supported frequencies.

Supported mappings (Examples are given for a mixed frequency VAR with monthly and quarterly data):

- 'levelSummed' : $Y(q) = Y(m) + Y(m-1) + Y(m-2)$
- 'diffSummed' : $Y(q) = Y(m) + 2*Y(m-1) + 3*Y(m-2) + 2*Y(m-3) + Y(m-4)$
- 'levelAverage' : $Y(q) = 1/3*(Y(m) + Y(m-1) + Y(m-2))$
- 'diffAverage' : $Y(q) = 1/3*Y(m) + 2/3*Y(m-1) + Y(m-2) + 2/3*Y(m-3) + 1/3*Y(m-4)$
- 'end' : $Y(q) = Y(m)$

The default mapping that is used for all series with lower frequency is 'diffAverage'.

Input:

- obj : An object of class nb_mfvar
- map : A cellstr on the format; {'Var1','levelSummed',...
'Var2','diffSummed'}

Output:

- obj : An object of class nb_mfvar where the mapping of a set of variables has been set. See the dependent and block_exogenous properties.

See also:

`nb_mfvar.setFrequency`

Written by Kenneth Sæterhagen Paulsen

► **setMeasurementEqRestriction** ↑

`obj = setMeasurementEqRestriction(obj, restrictions)`

Description:

Add restrictions in the measurement equation of the model on the form;

`restricted = parameters*variables'`

e.g.

`X = [0.5,0.5]*[Y;Z]`

where X is a observed variable, while Y and Z are state variables.
So for a mixed frequency var, X and Y must be included in the state equation. An error is provided if this is not the case!

For the xxxMF priors this information is taken into account, otherwise it will only be used during conditional forecasting, i.e. the measurement restrictions will be appended to the solution after estimation is done.

See `nb_var.priorTemplate` for more on the supported priors.

Input:

- obj : An object of class nb_var
- restrictions : A struct array with fields
 - > 'restricted' : A one line char with the variable to restrict. It must be a dependent variable or block exogenous variable.
 - > 'parameters' : A cellstr with the names of the variables storing the parameters of the

restriction, or a double array. Must have same length as 'variables'.
 > 'variables' : A cellstr with the names of the variables included in the restriction.
 > 'frequency' : Set to the frequency of the observables. This is only important for the mixed frequency VAR model. Give [], if the restriction is on the same frequency as the data. Either 1 (yearly), 4 (quarterly) or 12 (monthly).
 > 'mapping' : Sets the mapping to use for this restricted variable. See the method nb_mfvar.setMapping for more on the mappings to choose from.
 > 'R_scale' : Inverse prior scale of the variance of the measurement error of this restrictions.

The variance of the measurement error is constructed by dividing the variance of each 'restricted' variable by this scaling parameter.

Output:

- obj : An object of class nb_var where the measurement equation restrictions are added.

See also:

[nb_mfvar.template](#), [nb_bVarEstimator.applyMeasurementEqRestriction](#)
[nb_mfvar.priorTemplate](#), [nb_mfvar.setMapping](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_VAR

► **setMixing** ↑

obj = setMixing(obj,mixing)

Description:

Set the mixing of the decleared dependent and/or block_exogenous variables of a formulated model, i.e. if some of the dependent variables represents the same variable, but with different frequencies.

See [setFrequency](#) for the supported frequencies.

Input:

- obj : An object of class nb_mfvar
- mixing : A cellstr on the format; {'Var1_Q','Var1_M',... 'Var2_Q','Var2_M'}. For more see [nb_mfvar.help\('mixing'\)](#).

Output:

- obj : An object of class nb_mfvar where the mixing of a set of variables has been set. See the dependent and block_exogenous properties.

See also:

[nb_mfvar.setFrequency](#)

Written by Kenneth Sæterhagen Paulsen

► **setPrior** ↑

obj = setPrior(obj,prior)

Description:

Set priors of a vector of nb_var objects. The prior input must either be a struct or nb_struct with same size as obj or have size 1.

The provided struct will be added to the property options as the field prior.

See nb_var.priorTemplate for the format of the struct.

Caution : The estim_method field of the option property will automatically be set to 'bvar'.

Caution : Not yet supported for the subclass nb_favar!

Input:

- obj : A vector of nb_var objects
- prior : A struct or nb_struct with either matching numel of obj or size 1. If size is 1 all models gets the same prior settings.

Output:

- obj : A vector of nb_var objects with the priors set.

See also:

[nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **set_identification** ↑

```
obj = set_identification(obj,type,varargin)
```

Description:

Identify a VAR model. Either using cholesky restrictions or combinations of sign and zero restrictions (short, mid and long term).

The method solve must be called after adding the identifying restrictions

See Andrew Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". This code is an adaption of his code to the NBToolbox, except that also magnitude restriction is possible using this code.

Input:

- obj : A vector of nb_var objects, or subclasses of this class.
- type : Either 'cholesky' or 'combination'
- varargin (Optional inputs) :
 - > 'cholesky' :
 - 'ordering' : A cellstr with the ordering of the identification. Default is to use the ordering of the dependent variables of the model. The ordering is so that the first variable is the variable that is only influenced by one shock in period 1, the second variable is the variable that is influenced by two shock in period 1, and so on.
 - Caution : All variables of the VAR must be included in this option.
 - > 'combination' :
 - 'maxDraws' : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.
 - 'draws' : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Caution : When it comes to producing IRFs with sign restrictions you have two options. The first is to set 'draws' to a large number, say 1000, and then produce irf for each of those draws. The second approach is to draw parameters using bootstrap, MC methods or from the posterior, and then to make one draw of the C for each draw of the paramters.

```

- 'restrictions' : A nRestriction x 4 cell;
    {Dep,Shock,period,type,value;...}

    > Dep      : The dependent variable to add the
                  restriction to. As a string.

    > Shock    : The name of the identified structural
                  shock. As a string.

    > period   : The period of the restriction, as a
                  double. For on impact set 0 and for
                  cumulative restriction set it to inf.

    Use s:e to add a restriction on the
    cumulative contribution of a shock
    to a variable starting at s and
    ending at e. E.g. 0:10.

    > type     : Either 0 for zero restriction,
                  '+/-' for sign restrictions or
                  '>/<' for magnitude restrictions.

    Or 'none' : If you want to add a
                structural shock with no restriction,
                as you already have exactly
                N-1 shocks.

    > value    : A 1x1 double with the value of the
                  magnitude restriction.

```

Output:

- obj : Identified nb_var objects. See the property field identification of the property solution.

Examples:

See ...\\Econometrics\\test\\test_nb_var

Written by Kenneth Återhagen Paulsen
Subfunctions are written by Andrew Binning 2013

Inherited from superclass NB_VAR

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the

method nb_var.set_identification for more.

- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_mfvar.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [simulate ↑](#)

out = simulate(obj,nSteps,varargin)

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
 - 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e.

each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class

nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

> double : A 1 x nVar double.

> 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsge models.

> 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'startingProb' : Either a double or a char:

> double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments ↑**

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.
- Optional inputs;
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find

```
cov(x,y(-i)) you can use  
getVariable(varargin{i},'y','x'),  
while to get cov(x(-i),y) (== cov(x,y(+i))) you  
can use getVariable(varargin{i},'x','y'). (This  
example only works if the output is of class nb_cs)
```

See also:

[nb_mfvar.graphCorrelation](#), [nb_mfvar.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve** ↑

```
obj = solve(obj)
```

Description:

Solve estimated model(s) represented by nb_mfvar object(s).

Input:

- obj : A vector of nb_mfvar objects.

Output:

- obj : A vector of nb_mfvar objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_mfvar.solveNormal(results,opt,ident)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_mfvar.solveRecursive(results,opt,ident)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **stateSpace** ↑

```
[x0,P0,d,H,R,T,c,A,B,Q,G,obs,failed] = nb_mfvar.stateSpace(par,nDep,...  
nLags,nExo,restr,restrVal,measErrInd,...  
H,stabilityTest)
```

Description:

Returns a state-space representation of an MF-VARX(nLags) model.

Observation equation:

$y = d + Hx + Tz + v, \quad v \sim N(0, R)$

State equation:

$x = c + Ax_{-1} + Gz + Bu, \quad u \sim N(0, I)$

Input:

- par : The parameter vector of the VARX model.

Order:

- constant
- exogenous
- lagged endogenous
- residual std
- measurement error std

- nDep : The number of dependent variables of the model.
- nLags : The number of lags of the VAR (of the highest frequency).

Caution : The state-space representation may be bigger due the measurement equation.
- constant : Include constant or not. true or false.
- nExo : Number of exogenous variables included in the model.
- restr : A nDep x nDep*nLags logical matrix. true fro the non-restricted parameters.
- restrVal : Values of the restricted parameters.
- measErrInd : The index of the measurement error std to estimate. If empty no measurement errors are allowed.
- H : The matrix of the measurement equation.
- stabilityTest : Check stability of system. Sets failed to true if model is not stationary. Default is false.

Output:

- x0 : Initial state vector.
- P0 : Initial variance.
- See equation above
- obs : Index of observables in the state vector.
- failed : See stabilityTest input.

See also:

[nb_kalmanlikelihood_missing](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **struct ↑**

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_mfvar.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_mfvar.template()
options = nb_mfvar.template(num)
```

Description:

Construct a struct which can be provided to the nb_mfvar class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

```

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
    {variable, date, test}.

    > variable : The variable(s) to test,
        as a string or cell array of strings.

    > date : The date as a string. If a cell array is given, a
        copy of the restriction will be made for every given
        date.

    > test : Restriction test as a function handle. The value(s)
        of the variable(s) in 'variable' at time 'date' are
        passed as arguments. Should return a logical.

```

Output:

- out : A double with the probability

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
    {'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);

```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```

- obj : A scalar solved nb_model_generic object.

- expression : A MATLAB expression as a string. Use parameter names as
    variables. See model.parameters.name.

```

```

- method      : A string with the method to use. For bootstrap method see
               nb_bootstrap. For bayesian models 'posterior' is the only
               option. Default is 'bootstrap' for classical models, while
               'posterior' is the default for bayesian models.

- draws       : Number of draws from the parameter distribution. Default
               is 1000.

- alpha       : Confidence level. Default is 0.05.

- type        : Type of test:
               > '=' : Two sided test. expression == 0 (default)
               For two-sided tests it is assumed that the
               distribution is symmetric! Use
               confidenc/probabillty intervals instead.
               > '>' : One sided test. expression > 0
               > '<' : One sided test. expression < 0

```

Output:

```

- pval        : > 'classical' : P-value of test.
               > 'bayesian'  : Probabilty of test.

               A 1x1 double.

- ci          : A 1 x 2 double with the lower and upper bound of the
               confidence/probabillty interval of the tested expression.

- dist        : A nb_distribution object storing the distribution of the
               tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► **theoreticalMoments** ↑

```

[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)

```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following
way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.
- Optional inputs;
- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.
Can also be a cellstr, with a subset of variables from 'full'.
- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_mfvar.graphCorrelation](#), [nb_mfvar.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► uncertaintyDecomposition ↑

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the

percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).

> 'fev'	: Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc'	: At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages'	: Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_mfvar.forecast](#)

Written by Kenneth Åstterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_mfvar.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update](#) ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► variance_decomposition ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
> 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent

variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.

- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.

- `plotter` : A 1 x nModel vector of nb_graph_cs objects. Use the `graph` method or the nb_graphPagesGUI class to produce the graphs.
- `plotterBands`: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_mfvar.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_midas

Go to: [Properties](#) | [Methods](#)

A class for estimation of MIDAS (MIxed frequency DAta Sampling) models.

$y(tq+hq) = \lambda y(tq) + F(x(tm), x(tm-1), \dots, p) + e$

Where $x(tm)$ are lagged exogenous variables of the model at high frequency, and y is the dependent variable of the model.

Superclasses:

[nb_model_generic](#)

Constructor:

`obj = nb_midas(varargin)`

Optional input:

- See the `set` method for more on the inputs.
([nb_model_estimate.set](#))

Output:

- `obj` : An nb_midas object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_midas.help](#), [nb_midas.template](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- | | | | | |
|----------------------------|-----------------------------------|--------------------------|-------------------------------|--------------------------|
| • <code>addAutoName</code> | • <code>addAutoNameIfLocal</code> | • <code>addID</code> | • <code>addIDIfLocal</code> | • <code>dependent</code> |
| • <code>endogenous</code> | • <code>estOptions</code> | • <code>exogenous</code> | • <code>forecastOutput</code> | • <code>name</code> |
| • <code>options</code> | • <code>parameters</code> | • <code>reporting</code> | • <code>residuals</code> | • <code>results</code> |
| • <code>solution</code> | • <code>transformations</code> | • <code>userData</code> | | |

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input

variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.

- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.

- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.

- F : See the equation above. A nobs x nExo double. (Storing the impact of exogenous terms of the observation equation)

- G : See the equation above. A nobs x nfact double.

- observables : A 1 x nobs cellstr with the declared observable variables.

- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.

- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use;

methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** [↑](#)

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- `getScore`
- `getVariablesList`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`
- `removeObservations`
- `setFrequency`
- `simulate`
- `simulatedMoments`
- `solveNormal`
- `solveVector`
- `template`
- `testParameters`
- `uncertaintyDecomposition`
- `unstruct`
- `variance_decomposition`
- `getSolution`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `irf`
- `isFiltered`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `set`
- `shock_decomposition`
- `simulateFromDensity`
- `solve`
- `solveRecursive`
- `struct`
- `testForecastRestrictions`
- `theoreticalMoments`
- `uncondForecast`
- `update`

► **`appendData`** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_midas.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods. Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convert`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convert`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

`obj = convert(obj,freq,methods,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_midas.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_midas.forecast](#), [nb_midas.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_midas.forecast`, `nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast`, `nb_midas.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_midas.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj          = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'      : The length of the bins og the domain of the density.
  Either;
    > []       : The domain will be found. See
      nb_getDensityPoints (default is that 1000
      observations of the density is stored).
    > integer   : The min and max is found but the length
      of the bins is given by the provided
      integer.
    > cell       : Must be on the format:
      {'Var1',lowerLimit1,upperLimit1,binsL1;
       'Var2',lowerLimit2,upperLimit2,binsL2;
       ...}
    - lowerLimit1 : An integer, can be nan,
      i.e. will be found.
    - upperLimit1 : An integer, can be nan,
      i.e. will be found.
    - binsL1      : An integer with the
      length of the bins. Can
      be nan. I.e. bins length
      will be adjusted to
      create a domain of 1000
      elements.

  Caution : The variables not included will be given
            the default domain. No warning will be
            given if a variable is provided in the
            cell but not forecast by the model. (This
            because different models can forecast
            different variables)

  Caution : The combineForecast method of the
            nb_model_group class is much faster if the
            lower limit, upper limit! Or else
            it must simulate new draws and do kernel
            density estimation for each model again
            with the shared domain of all models
            densities.

- 'startDate'   : The start date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

  Caution: If 'fcstEval' is not empty this will set
            the start date of the recursive forecast.
            Default is to start from the first possible
            date.

- 'endDate'     : The end date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

  Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
 the ergodic steady-state for
 MS-model, and in the last regime
 for break-point models.

- 'startingProb' : Either a double or a char:
 > 'zero' : Start from zero on all variables
 (Except for the deterministic
 exogenous variables)

> [] : If the model is filtered the
 starting value is calculated
 on filtered transition
 probabilities. Otherwise the
 ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x
 nRegime double. nPeriods refer to
 the number of recursive periods to
 forecast.

> 'ergodic' : Start from the ergodic transition
 probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws
 that give rise to an unstable model. Default
 is false.

- 'states' : Either an integer with the regime to
 forecast/simulate in, or an object of class nb_ts
 with the regimes to condition on. The name of the
 variable must be 'states'.

Caution: For break-point models this is decided by
 the dates of the breaks, and cannot be set
 by this option!

- 'compareToRev' : Which revision to compare to. Default is final
 revision, i.e. []. Give 5 to get fifth revision.
 must be an integer. Keep in mind that this number
 should be larger than the nSteps input.

- 'compareTo' : Sometime you want to evaluate the forecast of one
 variable against another variable which is not
 part of the model. E.g. if you use the first release
 of a variable and want to compare it against the
 final release. To do this you can give a cellstr
 with size n x 2, where n is the number of variables
 to change the the actual observations to test
 against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
 kernel density estimation of the density
 forecast.

- 'exoProj' : Tolerate missing exogenous variables by projecting
 them by a fitted model. Only when the 'condDB'
 input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
- 'exProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.
- 'exProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.
- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.
- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_midas.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_midas.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_midas.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_midas.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_midas.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_midas.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getForecastVariables ↑](#)

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_midas object, subclass of the nb_model_generic class.
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables is found.
- date : For recursive real-time estimated models the history may change. This will give the correct vintage for a given recursion.
- notSmoothed : Just for generics. Not in use for the nb_midas class.
- type : Just for generics. Not in use for the nb_midas class.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```

pit                  = getPIT(obj)
[pit,plotter]       = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)

```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

nb_calculatePIT

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the

```
parameter values ('struct' is not supported if
numel(obj) > 1).

: Give 'headers' to add model names as headers for each
column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.
```

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base
the kernel estimation on, if empty all draws are used for
estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.

- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted ↑**

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph ↑**

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

residualNames = getResidualNames(obj)

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

roots = getRoots(obj)

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursivly estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
 - type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- If the object input is a vector, the type input can also be a cellstr array with matching length.
- Caution: See comment to the invert input!
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
 - startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
 - endDate : The date to end constructiong the score. Can be

empty. Either as a string or an object of class nb_date.

- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_midas.simulatedMoments](#), [nb_midas.theoreticalMoments](#)
[nb_midas.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_midas.help
helpText = nb_midas.help(option)
helpText = nb_midas.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize ↑**

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given,

otherwise this option does not apply.

- 'optimizer'
 - : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'
 - : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'
 - : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'
 - : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods'
 - : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'
 - : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'
 - : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'
 - : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'
 - : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights'
 - : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_midas.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgc objects.
- 'compare' : Give true if you have a scalar nb_dsgc model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

```

- 'draws' : Number of draws for calculating girf. Default is
           1000. For Markov switching models this will set the
           number of simulated paths of states. For more see
           the 'type' input.

- 'factor' : A cell array with the factors to multiply the
             irf of the individual model variables.
             I.e. {'var1',100,...} or
                  {'var1','var2'},100,...}

If the string starts with a asterisk you will
multiply all the variables that contain that
string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This option sets the error band percentiles of the
               graph, when the 'perc' input is empty. As a 1 x
               numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
               Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from
                  this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

    > 'cumulative product' : cumprod(1 + X,1)

    > 'cumulative sum' : cumsum(X,1)

    > 'cumulative product (log)' : log(cumprod(1 + X,1))

    > 'cumulative sum (exponential)' : exp(cumsum(X,1))

    > 'cumulative product (%)' : cumprod(1 + X/100,1)

    > 'cumulative sum (%)' : cumsum(X/100,1)

    > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))

    > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))

    > '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
             Default is ''. See help on the method input of the
             nb_model_generic.parameterDraws method for more
             this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
                    the map from structural shocks to dependent

```

variables. See `nb_var.set_identification`. Of course this option only makes sense for VARs identified with sign restrictions.

- `'normalize'` : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to `inf`, it will normalize the max impact period.
E.g. `{'Var',1,2}, {'Var',1,inf}`
Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- `'normalizeTo'` : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- `'newReplic'` : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- `'parallel'` : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- `'parallelL'` : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.
- `'pause'` : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- `'perc'` : Error band percentiles. As a 1 x `numErrorBand` double. The input must be given as the coverage, and not the percentiles itself. E.g. `[0.3,0.5,0.7,0.9]`. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So `[0.3,0.5,0.7,0.9]` will get the percentiles `[5,15,25,35,65,75,85,95]`, i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- `'periods'` : Number of periods of the impulse responses.
- `'plotSS'` : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.

```

- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be
a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws
that give rise to an unstable model. Default
is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar : Select the the regime to take the
initial transition probabilities
from.

    > double : A draws x nRegime or a 1 x
nRegime double. draws refer to
the number set by the 'draws'
input.

    > 'ergodic' : Start from the ergodic transition
probabilities. Default for 'irf'.

    > 'random' : Randomize the starting values
using the simulated starting
points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing

```

with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

```

> 'zero'          : Start from zero on all variables.

> 'random'        : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'         : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'           : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'   : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'       : The variables to create impulse responses of.
                     Default is the dependent variables defined by the
                     first model.

Caution: The variables reported by the reporting
property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                     using the method specified by 'levelMethod'.

```

Output:

```

- irfs      : The (central) impulse response function stored in a
               structure of nb_ts object. Each field stores the the
               impulse responses of each shock. Where the variables
               of the nb_ts object is the impulse responses of the
               wanted endogenous variables.

               I.e. the impuls responses to the shock 'E_X_NW' is
               stored in irfs.('E_X_NW'). Which will then be a nb_ts
               object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
responses from each model are saved as
different datasets (pages) of the nb_ts object.

```

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_midas.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic ↑**

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

► **isbayesian ↑**

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

ret = isestimated(obj)

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint

```
prediction bands for  
macroeconomic risk management  
  
> 'copula' : Using a copula likelihood approach.  
  
> 'mostlik' : Group the paths based on how  
likely they are.  
  
- 'nSteps' : Number of forecasting steps to use when constructing the  
error bands. If empty (default) all forecasting steps stored  
in the object is used.
```

Output:

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.
- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend
on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_midas.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. @(x)gt(x,0), @(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. @(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1), i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to

test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:

1. The expression using any of the endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_midas.monteCarloFiltering](#), [nb_midas.irf](#)
[nb_midas.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_midas.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.

- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_midas.mcfRestriction](#), [nb_midas.solve](#)
[nb_midas.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'copulaBootstrap' : Uses a copula approach to draw residual that are autocorrelated, Does not handle heteroscedasticity. Only an option for models estimated with classical methods.
 - > 'posterior' : Posterior draws. Only for models estimated with bayesian methods.

Default for models estimated with
bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more on this option.
- stable : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
- lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
- factors : Draws of the factors as a T x nFac x

```

draws double.

> 'solution' : Get the solution of the model for each draw from the
distribution of parameters. The output will be a struct
with size 1 x draws. The struct will have the same
format as the solution property of the underlying
object. I.e. only one output!

> 'object'   : Get the draws as a 1 x draws nb_model_generic object.
Each element will be a model representing a given draw
from the distribution of the parameters. I.e. only one
output!

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + ( +/- nb_distribution.t_icdf(alpha/2) ) * stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```

plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)

```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► plotMCF ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot.

'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- `plotter` : > 'allInOne' : A `nb_graph_cs` object. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
 > 'biplot' : A vector of `nb_graph_data` objects with size equal to the number of pairwise combination of the parameters that can be made. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_midas.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFDistTest ↑**

`plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)`

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `test` : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_midas.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `nameValue` : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_midas.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_midas.getPosteriorDistributions](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_midas.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties
of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setFrequency** ↑

obj = setFrequency(obj, freq)

Description:

Set the frequency of the decleared dependent and/or exogenous
variables of a formulated model.

Supported frequencies:

- 'yearly' : 1
- 'quarterly' : 4
- 'monthly' : 12
- 'weekly' : 52

Input:

- obj : An object of class nb_midas.

- freq : A scalar integer with the frequency of the dependent variable,
or a cellstr on the format; {'Var1',1,'Var2',4}. There is no
need to set the frequency of those variables that has the same
frequency as the data (options.data)!

Caution: The frequency of the dependent variable must be greater
or equal to the frequency of the exogenous!

Output:

- obj : An object of class nb_midas where the frequency of a set of
variables has been set. See the dependent and exogenous
properties.

Written by Kenneth Sæterhagen Paulsen

► **shock_decomposition** ↑

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';  
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the

nb_model_generic.parameterDraws method for more this input. An extra option is:

```
> 'identDraws' : Uses the draws of the matrix with  
      the map from structural shocks to dependent variables.  
      See nb_var.set_identification. Of course this option  
      only makes sense for VARs identified with sign  
      restrictions.  
  
- 'replic'          : The number of simulation for posterior, bootstrap and  
      MC methods. Default is 1. Only used when 'perc' is  
      provided.  
  
      Caution: Will not have anything to say for the method  
      'identDraws'. See the option 'draws' of the  
      method nb_var.set_identification for more.  
  
- 'stabilityTest' : Give true if you want to discard the draws  
      that give rise to an unstable model. Default  
      is false.  
  
- 'parallel'        : Give true if you want to do the shock decomp in  
      parallel. Default is false.  
  
- 'fcstDB'          : An object of class nb_ts with the forecast to  
      decompose. must contain all model variables and  
      shocks/residuals, and must start the period after  
      the estimation/filtering end date or at the  
      estimation/filtering start date. See  
      the nb_model_generic.getForecast method.  
  
- 'type'            : Choose 'updated' or 'smoothed'. 'smoothed' is  
      default.  
  
- 'anticipationStartDate' : Start date of anticipating shocks. As a  
      string. If different models has different  
      frequency this date will be converted.  
  
- 'model2'          : A struct with the solution of the second  
      model to use for decomposition.  
  
- 'secondModelStartDate' : Start date of second model. As a string.  
      If different models has different frequency  
      this date will be converted.
```

Output:

- decomp : A structure of nb_ts objects with the shock
 decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty
 bounds of the shock decomposition for each model at each
 percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the
 graph method to produce the graphs or use the
 nb_graphPagesGUI on each element of the graph objects.

See also:

`nb_midas.parameterDraws, nb_shockDecomp`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **simulate** ↑

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `nSteps` : Number of simulation steps. As a `1x1 double`. Default is 100.

Optional inputs:

- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- `'bounds'` : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - `'shock'` : Name of the shock to match the restriction on the bounds of the given variable.
 - `'lower'` : The lower bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.
 - `'upper'` : The upper bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps double` matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a `1x1 double` or a `nSteps x 1 double`.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty

the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadyState' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsg models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consists of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the

'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.
 Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
 Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
 Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i},'y','x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i},'x','y'). (This example only works if the output is of class nb_cs)

See also:

[nb_midas.graphCorrelation](#), [nb_midas.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_midas object(s).

Input:

- obj : A vector of nb_midas objects.

Output:

- obj : A vector of nb_midas objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_midas.solveNormal(results,opt)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_midas.solveRecursive(results,opt)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_midas.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_midas.template()
options = nb_midas.template(num)
```

Description:

Construct a struct which can be provided to the nb_midas class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

```

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
    {variable, date, test}.

    > variable : The variable(s) to test,
        as a string or cell array of strings.

    > date : The date as a string. If a cell array is given, a
        copy of the restriction will be made for every given
        date.

    > test : Restriction test as a function handle. The value(s)
        of the variable(s) in 'variable' at time 'date' are
        passed as arguments. Should return a logical.

```

Output:

- out : A double with the probability

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
    {'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);

```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```

- obj : A scalar solved nb_model_generic object.

- expression : A MATLAB expression as a string. Use parameter names as
    variables. See model.parameters.name.

```

```

- method      : A string with the method to use. For bootstrap method see
               nb_bootstrap. For bayesian models 'posterior' is the only
               option. Default is 'bootstrap' for classical models, while
               'posterior' is the default for bayesian models.

- draws       : Number of draws from the parameter distribution. Default
               is 1000.

- alpha       : Confidence level. Default is 0.05.

- type        : Type of test:
    > '=' : Two sided test. expression == 0 (default)
            For two-sided tests it is assumed that the
            distribution is symmetric! Use
            confidenc/probabillty intervals instead.
    > '>' : One sided test. expression > 0
    > '<' : One sided test. expression < 0

```

Output:

```

- pval        : > 'classical' : P-value of test.
               > 'bayesian' : Probabilty of test.

               A 1x1 double.

- ci          : A 1 x 2 double with the lower and upper bound of the
               confidence/probabillty interval of the tested expression.

- dist        : A nb_distribution object storing the distribution of the
               tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► **theoreticalMoments** ↑

```

[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)

```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following
way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.
- Optional inputs;
- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.
Can also be a cellstr, with a subset of variables from 'full'.
- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_midas.graphCorrelation](#), [nb_midas.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition** ↑

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the

percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).

> 'fev'	: Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc'	: At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages'	: Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_midas.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_midas.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update ↑](#)

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► variance_decomposition ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
> 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent

variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.

- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.

- `plotter` : A $1 \times nModel$ vector of `nb_graph_cs` objects. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
- `plotterBands`: A $1 \times nModel$ struct. Each field is a $1 \times nVars$ vector of `nb_graph_cs` objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_midas.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

■ **nb_model_convert**

Go to: [Properties](#) | [Methods](#)

A class for converting the frequency of `nb_model_forecasts`. This makes it easier to compare or combine forecasts from objects with different frequency.

Superclasses:

`nb_model_forecast`, `nb_model_estimate`

Constructor:

`obj = nb_model_convert(model, varargin)`

Input:

- `model` : An object of class `nb_model_generic` with stored forecasts.

Optional input:

- `'freq'` : The frequency you want to convert to. As integer. Either 1, 2, 4, 12 or 365.
- `'method'` : The method which you would like to use to convert the forecasts with. For more, see the `nb_ts.convert` method.
- `'name'` : To set a name for the object.
- `'interpolatedate'` : Use this to determine if you would like to choose the start of the new period or end of the new period as date as your forecast date. E.g. if you are going from quarterly to monthly and choose `'start'`, then 2010Q2 becomes 2010M4. Either `'start'` or `'end'`, default is `'start'`.

Output:

- obj: An object of class nb_model_convert

Examples:

- nb_model_convert(obj,'freq',12,'method','linear','name','test',...
 'interpolatedate','start')

See also:

[nb_model_generic](#)

Written by Tobias Ingebrigtsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [estOptions](#)
- [forecastOutput](#)
- [model](#)
- [name](#)
- [options](#)
- [reporting](#)
- [results](#)
- [transformations](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[estOptions](#)** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **[forecastOutput](#)** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **model** ↑

The nb_model_generic object with the original forecast

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that persevere the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that persevere the time span are supported in expression. You can also call user defined

functions, as long as they take and return a nb_math_ts object.
A nb_math_ts object is just a double with a time dimension!
To get a list of all the methods of the nb_math_ts class use;
`methods('nb_math_ts')`.

Can also be empty.

See also: `nb_ts.createVariable` and `nb_ts.createShift`.

Inherited from superclass `NB_MODELDATA`

Methods:

- `appendData`
- `checkModel`
- `checkReporting`
- `convert`
- `convertEach`
- `convertModel`
- `createVariables`
- `doForecastPerc2Dist`
- `doForecastPerc2ParamDist`
- `doSimulateFromDensity`
- `estimate`
- `evalFcstAtDates`
- `evaluateForecast`
- `forecast`
- `forecastPerc2Dist`
- `forecastPerc2ParamDist`
- `getDependentNames`
- `getEstimationOptions`
- `getEstimationStartDate`
- `getForecast`
- `getForecastVariables`
- `getHistory`
- `getModelNames`
- `getOriginalVariables`
- `getPIT`
- `getRecursiveScore`
- `getResidualNames`
- `getScore`
- `handleCondInfo`
- `handleMissing`
- `help`
- `isDensityForecast`
- `isForecasted`
- `isStatic`
- `mincerZarnowitzTest`
- `plotForecast`
- `plotForecastDensity`
- `print`
- `print_estimation_results`
- `removeObservations`
- `set`
- `simulateFromDensity`
- `struct`
- `template`
- `testForecastRestrictions`
- `uncondForecast`
- `unstruct`
- `update`

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **checkModel** ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_convert.

Output:

- obj : An object of class nb_model_convert.

Written by Kenneth SÃ¶terhagen Paulsen

► checkReporting ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODELDATA

► convert ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertModel** ↑

```
obj = convertModel(obj)
```

Description:

A method for converting the frequency of an nb_model_convert object with forecasts.

Caution: When converting from high frequency to lowe frequency it uses the same information content, as is the case for the last forecast produced, for all recursive periods. This means that if 2000Q2 is the last forecast start date, then it will fetch the forecast from 1999Q2, 1998Q2, etc...

Input:

- obj : An object of class nb_model_convert with stored forecasts.

Output:

- obj : An object of class nb_model_convert with the converted forecasts.

Written by Tobias Ingebrigtsen

► createVariables ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the nb_ts.createVariable method.

> third column : Any expression that can be interpreted by the nb_ts.createShift method.

> fourth column : Comments

- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,   description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **doForecastPerc2Dist** ↑

```
obj = doForecastPerc2Dist(obj, draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_convert.forecast](#), [nb_model_convert.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_convert.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity ↑**

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_model_generic.forecast`, `nb_model_convert.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **estimate** ↑

```
obj      = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by `nb_model_estimate` object(s).

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `'parallel'` : Use this string as one of the optional inputs to run the estimation in parallel.
- `'cores'` : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if `'parallel'` is given.
- `'remove'` : Give `'remove'` to remove the models that return errors from the `obj` output. This input is not stored to the `forecastOutput` property of the objects.
- `'waitbar'` : Use this string to give a waitbar during estimation. I.e. when looping over models. If `'parallel'` is used this option is not supported!
- `'write'` : Use this option to write errors to a file instead of throwing the error.

Optional input:

- `varargin` : See the the set method.

Output:

- `obj` : A vector of `nb_model_generic` objects, where the estimation results are stored in the property `results`.
- `valid` : A logical with size `nObj` x 1. true at location `ii` if estimation of model `ii` succeeded, otherwise false. If `'write'` is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

nb_model_generic, solve

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d,{'MSE'},0,1)
m = evalFcstAtDates(b,d,{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **forecast** ↑

```
obj = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_convert objects. First the model stored in the model property is forecast then the forecast is converted.

Input:

- obj : A vector of nb_model_convert objects.
- nSteps : See the forecast method of the nb_model_generic class

Optional inputs:

- See the forecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_convert objects.

Written by Kenneth Sæterhagen Paulsen

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_convert.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_convert.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_convert objects

Input:

- obj : A vector of nb_model_convert objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the model. The return value is the estimation start date of the model that is ordered last.

Input:

- obj : An object of class nb_model_convert

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!

Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_model_convert.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

```
- obj : An object of class nb_model_forecast.
```

Output:

```
- vars : A cellstr with the variables of the original data source.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory ↑**

```
histData = getHistory(obj,vars)
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data.

Input:

```
- obj : A scalar nb_model_convert object.
- vars : A cellstr with the variables to get. May include
          shocks/residuals. Only the variables found is returned,
          i.e. no error is provided if not all variables is found.
- date : Ignored
- notSmoothed : Ignored
- type : Ignored
```

Output:

```
- histData : A nb_ts object with the historical data.
```

Written by Kenneth SÃ¶terhagen Paulsen

► **getModelNames ↑**

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will
get the PIT from the start date of the recursive forecast and
use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct
the PITs based on the period from the estimation start date
until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getRecursiveScore ↑](#)

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getResidualNames ↑](#)

residualNames = getResidualNames(obj)

Description:

Get all the unique residual names of a vector of nb_model_convert object

Input:

- obj : A vector of nb_model_convert objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert, ...
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.

- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_model_convert.help
helpText = nb_model_convert.help(option)
helpText = nb_model_convert.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► mincerZarnowitzTest ↑

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

Input:

- obj : An object of class nb_model_group. You need to call the combineForecast method first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_model_generic.combineForecast](#)

Written by Kenneth SÃ¶terhagen Paulsen

► plotForecast ↑

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► print ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_convert.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object

- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **struct ↑**

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_convert.

Output:

- s : A struct representing the nb_model_convert object.

See also:

[nb_model_convert.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

► **template ↑**

```
options = nb_model_convert.template()
options = nb_model_convert.template(num)
```

Description:

Construct a struct which can be provided to the nb_model_convert class constructor.

This structure provided the user the possibility to set different conversion options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.

> variable : The variable(s) to test,
as a string or cell array of strings.

> date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.

> test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
                {'QSA_URR', 'QSA_DPQ_CP'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_convert objects.

Input:

- obj : A vector of nb_model_convert objects.
- nSteps : See the uncondForecast method of the nb_model_generic class

Optional inputs:

- See the uncondForecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_convert objects.

See also:

[nb_model_convert.forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

```
obj = nb_model_convert.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_convert class.

Input:

- s : A struct. See nb_model_convert.struct.

Output:

- obj : An object which is a subclass of the nb_model_convert class.

See also:

[nb_model_convert.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update ↑**

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of the object, and optionally re-estimate, re-solve, forecast the models given the updated data and convert the forecast.

Input:

- obj : A vector of objects of class nb_model_group
- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve
> 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.
- groupIndex : A 1 x nLevel double with the group level.

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if estimation or forecasting fails. The models that fail will then be removed when calling methods as compareForecast and aggregateForecast etc. See the valid property.

Output:

- obj : A vector of objects of class nb_model_convert.
- valid : A logical with size 1 x nObj. true at location ii if updating of model group ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

■ nb_model_estimate

Go to: [Properties](#) | [Methods](#)

An abstract superclass for all model objects that can be estimated.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- estOptions • options • reporting • results • transformations

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that pserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

Methods:

- [appendData](#)
- [convert](#)
- [createVariables](#)
- [getOriginalVariables](#)
- [handleMissing](#)
- [print](#)
- [removeObservations](#)
- [struct](#)
- [checkReporting](#)
- [convertEach](#)
- [estimate](#)
- [handleCondInfo](#)
- [isStatic](#)
- [print_estimation_results](#)
- [set](#)
- [unstruct](#)

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **convert** ↑

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables ↑**

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,   description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **estimate** ↑

```
obj          = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **handleCondInfo** ↑

```
ret = handleCondInfo(obj)
```

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

► **print_estimation_results ↑**

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations ↑**

```
obj = removeObservations(obj, numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► set ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.
Where you can set all fields of some properties
of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_model_estimate.unstruct](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **unstruct** ↑

obj = nb_model_estimate.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_model_estimate.struct](#)

Written by Kenneth Sæterhagen Paulsen

■ nb_model_generic

Go to: [Properties](#) | [Methods](#)

An abstract superclass of all model classes that is both estimated and forecasted.

Constructor:

No constructor exist. This class is abstract.

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|---------------|----------------------|-------------|------------------|-------------|
| • addAutoName | • addAutoNameIfLocal | • addID | • addIDIfLocal | • dependent |
| • endogenous | • estOptions | • exogenous | • forecastOutput | • name |
| • options | • parameters | • reporting | • residuals | • results |
| • solution | • transformations | • userData | | |

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **dependent** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t$, $e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t$, $u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the

model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- getScore

- getSolution
- graphCorrelation
- handleMissing
- interpretForecast
- isDensityForecast
- isMS
- isNB
- isStatic
- isestimated
- issolved
- loadPosterior
- mincerZarnowitzTest
- parameterDraws
- plotForecast
- plotMCF
- plotMCFValues
- plotPriors
- printCov
- removeObservations
- shock_decomposition
- simulateFromDensity
- solveVector
- testForecastRestrictions
- theoreticalMoments
- uncondForecast
- update
- getVariablesList
- handleCondInfo
- initialize
- irf
- isFiltered
- isMixedFrequency
- isStateSpaceModel
- isbayesian
- isforecasted
- jointPredictionBands
- mcfRestriction
- monteCarloFiltering
- parameterIntervals
- plotForecastDensity
- plotMCFDistTest
- plotPosteriors
- print
- print_estimation_results
- set
- simulate
- simulatedMoments
- struct
- testParameters
- uncertaintyDecomposition
- unstruct
- variance_decomposition

► **appendData** ↑

```
obj = appendData(obj, DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over

the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

obj = assignParameters(obj,varargin)

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

► **assignPosteriorDraws** ↑

```
obj = assignPosteriorDraws(obj,varargin)
```

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same

size as `obj.results.sigma`. The draws must come in the third dimension.

- `'lambda'` : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as `obj.results.lambda`. Only for factor models. The draws must come in the third dimension.
- `'R'` : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as `obj.results.R`. Only for factor models. The draws must come in the third dimension.
- `'period'` : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- `obj` : An object of class `nb_model_generic`, where the parameters of the underlying model has been changed. See `solve` to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

► **assignTexNames ↑**

```
obj = assignTexNames(obj, names, texNames)
```

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as `_t` is automatically appended to each variable! Use superscripts instead.

Input:

- `obj` : An object of class `nb_model_generic`.
- `names` : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- `texNames` : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- `obj` : A `nb_model_generic` object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its `tex_name` fields. (Also `observables` and `observablesFast` for factor model.)

See also:

`nb_dsge.writeTex`, `nb_dsge.writePDF`

Written by Kenneth Sæterhagen Paulsen

► **calculateMultipliers** ↑

```
mult = calculateMultipliers(obj,varargin)
```

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is `delta_Y` and `delta_G` not `Y` and `G` of the formula (3)!

Input:

- `obj` : An object of class `nb_dsge`.

Optional input:

- `'variables'` : A cellstr with the variables to produce the multipliers of. Must be provided!
- `'instrument'` : A one line char with the name of the instrument. Must be provided!
- `'rate'` : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- `'shocks'` : A cellstr with the shocks to produce the multipliers of. Must be provided!
- `'gross'` : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- `'perc'` : Error band percentiles. As a $1 \times nPerc$ double. E.g. $[0.3, 0.5, 0.7, 0.9]$. If empty all the simulation will be returned.
- `rest` : Use the `'replic'` option to produce error bands.

Output:

- `mult` : A $nShocks \times nVars \times nSim$ ($nPerc$) `nb_cs` object. Use the `double` method on this output to convert it to a double matrix.

See also:

[nb_model_generic.irf](#), [nb_calcMultiplier](#)

► **calculateStandardError** ↑

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel

```
density estimator of the parameter  
distribution. Two-sided test against the  
null hypothesis of the parameters being 0.  
  
- ci : Confidence interval/probability interval calculated based on  
a kernel density estimation instead of assuming asymptotic  
normal distribution. Only for the parameter in beta! As a  
(nCoeff * nEq) + 1 x 3 cell array.  
  
- dist : Estimated distribution of the coefficients of the main  
equation. As a (nCoeff * nEq) x 1 nb_distribution object.
```

Written by Kenneth Sæterhagen Paulsen

► **checkModel** ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options,
estOptions and results struct properties.

Input:

```
- obj : An object of class nb_model_generic.
```

Output:

```
- obj : An object of class nb_model_generic.
```

Written by Kenneth Sæterhagen Paulsen

► **checkPosteriors** ↑

```
[DB,plotter,pAutocorr] = checkPosteriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they were drawn. This to check for
problems with posterior draws that may be autocorrelated.

Input:

```
- obj : A scalar object of class nb_model_generic. It must represent  
a bayesian model that is estimated.
```

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot.
Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
 - plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
 - pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- Caution: Only plots the autocorrelation for the first chain
if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments** ↑

c = conditionalTheoreticalMoments(obj,varargin)

Description:

Calculate conditional theoretical moments; I.e.
 (auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
 Equivalent to the output of the theoreticalMoments method
 when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
 results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
 the moment of the dependent variables of the model without
 lag, while full will include all lags as well. 'dependent'
 is default.

Can also be a cellstr, with a subset of variables
 from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
 estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
 E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
 all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
 Default is ''. See below for what that means.

> 'bootstrap' : Create artificial data to bootstrap
 the estimated parameters. Default
 for models estimated with classical
 methods.

> 'wildBootstrap' : Create artificial data by wild
 bootstrap, and then "draw" parameters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'blockBootstrap' : Create artificial data by
 non-overlapping block bootstrap,
 and then "draw" parameters based
 on estimation on these data.
 Only an option for models estimated
 with classical methods.

```

> 'mBlockBootstrap'      : Create artificial data by
                           overlapping block bootstrap,
                           and then "draw" parameters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'rBlockBootstrap'      : Create artificial data by
                           overlapping random block length
                           bootstrap, and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'wildBlockBootstrap'   : Create artificial data by wild
                           overlapping random block length
                           bootstrap., and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'posterior'            : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

                           Caution : Posterior draws is
                           already made at the
                           estimation stage.

- 'nSteps'      : Number of steps ahead. As an integer.

```

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

► **constructCondDB** ↑

```

condDB           = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)

```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.

- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

► **constructScore ↑**

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.

```

        - 'MSE'      : One over mean squared error.
        - 'MAE'      : One over mean absolute error.
        - 'ME'       : Mean error.
        - 'STD'      : One over Standard error of the forecast
                       error.
        - 'ESLS'     : Exponential of the sum of the log scores.
        - 'EELS'     : Exponential of the mean of the log scores.
        - 'MLS'      : Mean log score.

- allPeriods      : true if you want the scores to be calculated for all
                    periods recursively, or else give false, i.e. only
                    calculate the score for the last period. False is
                    default.

- startDate       : The date to begin constructing the score. Can be
                    empty.

- endDate         : The date to end constructing the score. Can be
                    empty.

- nSteps          : Select the number of forecasting steps. Can be
                    empty.

- rollingWindow   : Set it to a number if the combination weights are to
                    be calculated using a rolling window. Default is
                    empty, i.e. to calculate the weights recursively using
                    the full history at each recursive step.

- lambda          : Give the value of the parameter of the exponential
                    decaying weights on past forecast errors when
                    constructing the score. If empty the weights on all
                    past forecast errors are equal.

```

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

► **convert ↑**

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both
the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending
cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,    description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest** ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method.

Default.

> 'a' : Andrews selection method. AR(1) specification.

- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_model_generic.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **doForecastPerc2Dist** ↑

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_model_generic.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

```
obj = doSimulateFromDensity(obj,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_model_generic.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments** ↑

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.

```
- 'demean'      : true (demean data during estimation of the  
autocovariance matrix), false (do not). Default is true.
```

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_model_generic.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

► [eq](#) ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sense they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind     = v(1) == v
```

See also:

`isequal`

Written by Kenneth Sæterhagen Paulsen

► **estimate** ↑

```
obj      = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by `nb_model_estimate` object(s).

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `'parallel'` : Use this string as one of the optional inputs to run the estimation in parallel.
- `'cores'` : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if `'parallel'` is given.
- `'remove'` : Give `'remove'` to remove the models that return errors from the `obj` output. This input is not stored to the `forecastOutput` property of the objects.
- `'waitbar'` : Use this string to give a waitbar during estimation. I.e. when looping over models. If `'parallel'` is used this option is not supported!
- `'write'` : Use this option to write errors to a file instead of throwing the error.

Optional input:

- `varargin` : See the the `set` method.

Output:

- `obj` : A vector of `nb_model_generic` objects, where the estimation results are stored in the property `results`.
- `valid` : A logical with size `nObj` x 1. true at location `ii` if estimation of model `ii` succeeded, otherwise false. If `'write'` is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_ESTIMATE`

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
    quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- `obj` : A `nb_model_generic` or `nb_model_group` object containing forecasts.
- `dates` : A cellstring containing the dates that the forecast should be evaluated at. Can also be a `nb_date` object.
- `type` : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- `density` : Set true if you have a density forecast.
- `quart` : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- `evaluation` : A `nb_data` object with the evaluation scores for each observation and variable.
- `errorDates` : Since it is not always the case that all of the dates are being used, `errorDates` returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d,{'MSE'},0,1)
m = evalFcstAtDates(b,d,{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' : - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
- 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states

input is providing the full path of regimes. Default is 1.

- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length of the bins is given by the provided integer.
 - > cell : Must be on the format:
 - {'Var1',lowerLimit1,upperLimit1,binsL1; 'Var2',lowerLimit2,upperLimit2,binsL2; ...}
 - lowerLimit1 : An integer, can be nan, i.e. will be found.
 - upperLimit1 : An integer, can be nan, i.e. will be found.
 - binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given the default domain. No warning will be

given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The combineForecast method of the nb_model_group class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.

- 'endDate' : The end date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is empty this input will have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).

- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.

- 'condDB' : One of the following:

> A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.

> A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.

> A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also

possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)

> A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the

conditional information. If an shock/residual is not included here, it means that it is not activated.

> StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

> Horizon : Anticipated horizon of the shocks/residual. 1 is default.

> Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.

- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.

- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.

> 'endo' : All the endogenous variables are returned.

> 'fullendo' : All the endogenous variables are returned included the lag variables.

> 'full' : All the variables are returned included the lag variables. Also

including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Including exogenous and shocks.

- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.

> double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.

> 'mean' : Start from the mean of the data observations. Default.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

> 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'startingProb' : Either a double or a char:

> [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'states' : Either an integer with the regime to forecast/simulate in, or an object of class nb_ts with the regimes to condition on. The name of the

variable must be 'states'.

Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!

- 'compareToRev' : Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
 - 'compareTo' : Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.
 - 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.
 - 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
- Options are:
- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
 - 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.
- Caution :** If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
- Caution :** If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.
- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1,'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matix. For the order of the variables see; obj.solution.endo. The

value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► forecastPerc2ParamDist ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provide recursive conditional information.

Caution : If the dataset is of type real-time, this function interprets the endDate input as the estimation end date for the last recursion. It also assumes that each vintage gives rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_model_generic.forecast](#)

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by
nb_var.priorTemplate('dsge').

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **getDependent** ↑

```
dependent = getDependent(obj)
```

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- start : An object of class nb_date.
```

Written by Kenneth Sæterhagen Paulsen

► **getFiltered ↑**

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

```
- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.
```

Optional input:

```
- 'reporting' : Either 'stored' or a N x 3 cell array on the format
of the reporting property. If 'stored' it will use
the reporting property stored in the object. If that
is empty, no reporting is done. Default is not to do
any reporting.
```

Output:

```
- data : A nb_ts object with size nobs x nvars included the filtered
variables.
```

See also:

[nb_dsge.filter](#), [nb_model_generic.estimate](#)

► **getForecast** ↑

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
```

```

        fcstPerc = fcstPercData.(fields{ii});
        plotter = nb_graph_ts(fcstMean);
        plotter.set('fanDatasets',fcstPerc);
        plotter.graph();

    end

> 'date'      : A string or a nb_date object with the date of the
                  wanted forecast. E.g. '2012Q1'. The fcstData will
                  be a nb_ts object with size nHor x nVar x nPerc + 1,
                  while fcstPercData will be empty. nPerc will then be
                  the number of percentiles/simualtions. Mean will be at
                  last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                  nPeriods + nHor x nVar x nHor nb_ts object. This
                  option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
                options for the 'date' and 'horizon' outputType.
                Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                  size nRec x nVars. While the fcstPercData output
                  will be a nb_ts object with size nRec x nVars x nSim.
                  You can set the horizon to return by the optional
                  input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_model_generic.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables** ↑

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.

- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sætherhagen Paulsen

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).
- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

parameters

Written by Kenneth Sæterhagen Paulsen

► **getPosteriorDistributions** ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth SÃ¶therhagen Paulsen

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...,
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object intput is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_model_generic.simulatedMoments](#), [nb_model_generic.theoreticalMoments](#)
[nb_model_generic.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sætherhagen Paulsen

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

ret = handleMissing(obj)

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► initialize ↑

```
obj = initialize(template)
```

Description:

Intialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be intialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth SÃ¶terhagen Paulsen

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the

- variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \logFunc$, where \logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
 - 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **irf ↑**

[irfs, irfsBands, plotter] = irf(obj, varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

```

> 'cumulative product' : cumprod(1 + X,1)
> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
      the map from structural shocks to dependent
      variables. See nb_var.set_identification. Of
      course this option only makes sense for VARs
      identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

```

- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.
- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

```

- 'startingProb'      : Either a double or a char (Only for Markov-switching
models):

    > scalar           : Select the the regime to take the
                           initial transition probabilities
                           from.

    > double            : A draws x nRegime or a 1 x
                           nRegime double. draws refer to
                           the number set by the 'draws'
                           input.

    > 'ergodic'         : Start from the ergodic transition
                           probabilities. Default for 'irf'.

    > 'random'          : Randomize the starting values
                           using the simulated starting
                           points. Default for 'girf'.

- 'startingValues'   : Either a double or a char:

    > double            : A nVar x 1 double.

    > 'steadystate'     : Start from the steady state. If you are dealing
                           with a MS-model or a break-point model you can
                           indicate the regime by 'steadystate(regime)'. E.g.
                           to start in regime 1 give; 'steadystate(1)'. For
                           MS - models the default is the ergodic mean
                           (default as long as 'girf' is not selected as
                           'type'), while for break-point models it is to
                           start from the first regime.

    > 'zero'             : Start from zero on all variables.

    > 'random'           : Randomize the starting values using the simulated
                           starting points. Default when 'type' set to
                           'girf'.

- 'states'            : Either an integer with the states to produce irf in,
                           or a double with the states to condition on. Must
                           have size periods x 1 in the last case. Applies to
                           both MS - models and break-point models.

- 'type'               : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf'             : Generlized impulse response function
                           calculated as the mean difference between
                           shocking the model with a one period shock
                           (1 std) and no shock at all. Use the 'draws'
                           input to set the number of simulations to use
                           to base the calculation on. This type of
                           irfs must be used for non-linear models.

    > 'irf'                : Standard irf for linear models. Calculated by
                           shocking the model with a one period shock
                           (1 std).

- 'variables'          : The variables to create impulse responses of.

```

Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_model_generic.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

► **isNB** ↑

ret = isNB(obj)

Description:

This will return true for all objects except nb_dsgc objects!

For nb_dsgc objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth SÃ¶terhagen Paulsen

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth SÃ¶terhagen Paulsen

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **jointPredictionBands** ↑

[bands,plotter] = jointPredictionBands(obj,varargin)

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

```

- 'vars'      : A cellstr with the variables to return
                 the calculated joint prediction bands of.

- 'perc'      : Error band percentiles. As a 1 x numErrorBand
                 double. E.g. [0.3,0.5,0.7,0.9] (default) or
                 0.9. Cannot be empty!

- 'date'      : If recursive forecast has been produced,
                 you can use this option to choose which of
                 the recursive forecast to construct the
                 joint prediction bands of. Default is empty,
                 i.e. use the last forecast.

- 'method'    : Choose between:
                 > 'kolsrud' : See Akram et al (2016), Joint
                               prediction bands for
                               macroeconomic risk management
                 > 'copula'   : Using a copula likelihood approach.
                 > 'mostlik' : Group the paths based on how
                               likely they are.

- 'nSteps'    : Number of forecasting steps to use when constructing the
                 error bands. If empty (default) all forecasting steps stored
                 in the object is used.

```

Output:

```

- JPB        : An nb_ts object storing the joint prediction bands.
                 The percentiles are stored as datasets. See the
                 dataNames property for which page represent which
                 percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.

- plotter    : A nb_graph_ts object. Use the graphSubPlots method
                 to produce the graphs.

```

Written by Kenneth Sætherhagen Paulsen

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

```
- obj       : An object of class nb_model_generic.
```

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_model_generic.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

► **mcfRestriction** ↑

f = mcfRestriction(obj,type,restriction)

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use @(x)f1(x)&&f2(x), where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. @(x)gt(x,0),
@(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. @(x,y)gt(x,y),
@(x,y)gt(x-y,0)||lt(x-y,1), i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.

as output.

> 'corr' : A N x 4 cell, where the elements of each row are:

1. Name of the first variable.
2. Name of the second variable.
3. Number of periods to lag the 2. variable.
4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:

1. The expression using any of the endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_model_generic.monteCarloFiltering](#), [nb_model_generic.irf](#)
[nb_model_generic.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

► **mincerZarnowitzTest ↑**

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_model_generic.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

► monteCarloFiltering ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to

```

'double'. Return true if the model fit a certain
criteria.

- 'method'      : The method used to draw the monte carlo simulated
                  parameters. See the method input to the nb_monteCarloSim
                  function for the supported values. Default is 'latin'.

- 'parallel'    : Give true to run the MCF in parallel. Default is false.

- 'parameters' : A 1 x N cellstr with the parameter names to be drawn
                  from. Must be part of model, and must be provided!

- 'lowerBound'  : A 1 x N double with the lower bound on the parameters of
                  interest.

- 'upperBound'  : A 1 x N double with the upper bound on the parameters of
                  interest.

- 'waitbar'     : true or false. Default is true.

- 'output'      : Either 'double' or 'logical'. Default is 'logical'.

- 'seed'         : Set the seed to use to draw the monte carlo simulated
                  parameter sets. Default is 1. Seed is set back to old
                  state after this method is called!

```

Output:

```

- paramD        : The draws made from the parameter space. As a draws x N
                  double. Use paramD(success,:) to get the accepted
                  draws.

- values         : A N x 1 logical/double. An element is true/not nan if
                  the model where solved and the test function returned
                  a value.

- solvingFailed : A N x 1 logical. An element is true if the model could
                  not be solved.

- objAcc        : A 1 x nAcc vector of nb_model_generic objects
                  representing the accepted models.

```

See also:

[function_handle](#), [nb_model_generic.mcfRestriction](#), [nb_model_generic.solve](#)
[nb_model_generic.assignParameters](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **parameterDraws** ↑

```

out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)

```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.

```

> 'copulaBootstrap'      : Uses a copula approach to draw residual
                           that are autocorrelated, Does not handle
                           heteroscedasticity. Only an option for
                           models estimated with classical methods.

> 'posterior'           : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
on this option.

- stable : Give true if you want to discard the parameter draws
that give rise to an unstable model. Default is false.

```

Optional input:

```

- 'parallel'      : Run in parallel. true or false.

- 'cores'          : Number of cores to use. Default is to use all cores
available.

- 'newDraws'       : When out of parameter draws, this factor decides how
many new draws are going to be made. Default is 0.1.
I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB
version later than R2017B this number will
be the number of new draws, and not the
factor!!! E.g. set it to 100. If it is given
as a number less than 1, it will by default
be set to 100.

- 'initialDraws'  : When drawing parameters this factor decides how many
many draws that are produced before solving the model.
Default is 1. I.e. it is equal to draws input. For
VAR models with underidentification it may be a good
idea to set this to a value > 1 as you may drop a
lot of parameters when identification fails. Default
is 1.

```

Output:

```

- out      : The output depend on the input output:

> 'param'   : Default. A struct with the following fields:

beta    : A nPar x nEq x draws double with the
estimated parameters.

sigma   : A nEq x nEq x draws double with the
estimated covariance matrix.

For factor models these fields are also returned:

lambda  : Estimated coefficients of the observation

```

equation. As a nFac x nObservables x draws double.

R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Sæterhagen Paulsen

► parameterIntervals ↑

ci = parameterIntervals(obj,alpha)

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

$\text{beta} + (+/-\text{nb_distribution.t_icdf}(\alpha/2)) * \text{stdBeta}$.

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

► plotForecast ↑

```

plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)

```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF** ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of

interest.

- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_model_generic.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

► **plotMCFDistTest ↑**

plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size $1 \times N$. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

`nb_model_generic.monteCarloFiltering`, `nb_cucconiTest`, `nb_smirnovTest`

Written by Kenneth SÃ¶terhagen Paulsen

► **plotMCFValues** ↑

`plotter = nb_model_generic.plotMCFValues(paramD, params, nameValue)`

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- `paramD` : A $\text{draws} \times N$ double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A $1 \times N$ cellstr with the names of the parameters.
- `values` : A $\text{draws} \times 1$ double with the values of the monte carlo filtering.
- `nameValue` : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size $1 \times N$. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

`nb_model_generic.monteCarloFiltering`

Written by Kenneth SÃ¶terhagen Paulsen

► **plotPosteriors** ↑

```
plotter = plotPosterior(obj)
plotter = plotPosterior(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_model_generic.getPosteriorDistributions](#)

Written by Kenneth Sætherhagen Paulsen

► **plotPriors ↑**

```
plotter = plotPriors(obj,varargin)
```

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optional input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sætherhagen Paulsen

► **print ↑**

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sætherhagen Paulsen

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_generic.print](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

[decomp, decompBand, plotter] = shock_decomposition(obj, varargin)

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

Caution: Two special identifiers can be used;

'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.

- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_model_generic.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting

values of the simulation. Default is 0.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
 - 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be

discared for all other types of models.

- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.
- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the

```

state by 'steadyState(state)'.
E.g. to start in state 1 give;
'stadyState(1)'. Default for
nb_dsgc models.

> 'zero'      : Start from zero on all variables
                  (Except for the deterministic
                   exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'startingProb' : Either a double or a char:

    > double       : A nPeriods x nRegime or a 1 x
                      nRegime double. nPeriods refer to
                      the number of recursive periods to
                      forecast.

    > 'ergodic'    : Start from the ergodic transition
                      probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

► **simulateFromDensity ↑**

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass [NB_MODEL_FORECAST](#)

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- `obj` : An object of class `nb_model_generic`.
- Optional inputs;
- `'output'` : Either '`nb_cs`' or '`double`'. Default is '`nb_cs`'.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when '`stacked`' is set to true.
- `'vars'` : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- `'type'` : Either '`covariance`' or '`correlation`'. Default is '`correlation`'.
- `'draws'` : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- `'pDraws'` : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- `'perc'` : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: '`pDraws`' or '`draws`' must be set to a number > 1.
- `'nSteps'` : Number of simulation steps. As a 1×1 double. Default is 100.

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Default is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_model_generic.graphCorrelation](#), [nb_model_generic.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

`obj = solveVector(obj)`

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_model_generic.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **testForecastRestrictions** ↑

probability = testForecastRestrictions(model, restrictions)

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
- > variable : The variable(s) to test,
as a string or cell array of strings.
- > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
- > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or
posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as

```

variables. See model.parameters.name.

- method      : A string with the method to use. For bootstrap method see
                nb_bootstrap. For bayesian models 'posterior' is the only
                option. Default is 'bootstrap' for classical models, while
                'posterior' is the default for bayesian models.

- draws       : Number of draws from the parameter distribution. Default
                is 1000.

- alpha       : Confidence level. Default is 0.05.

- type        : Type of test:
                > '=' : Two sided test. expression == 0 (default)
                        For two-sided tests it is assumed that the
                        distribution is symmetric! Use
                        confidenc/probabillty intervals instead.
                > '>' : One sided test. expression > 0
                > '<' : One sided test. expression < 0

```

Output:

```

- pval        : > 'classical' : P-value of test.
                > 'bayesian'   : Probabilty of test.

                A 1x1 double.

- ci          : A 1 x 2 double with the lower and upper bound of the
                confidence/probabillty interval of the tested expression.

- dist        : A nb_distribution object storing the distribution of the
                tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

► **theoreticalMoments ↑**

```
[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_model_generic.graphCorrelation](#), [nb_model_generic.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

► **uncertaintyDecomposition** ↑

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of

each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).

> 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.

- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.

- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd', 'fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth SÃ¶therhagen Paulsen

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_model_generic](#).struct

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update** ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failles. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if

estimation or forecasting of model is succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Åkerhagen Paulsen

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.

```

- 'output'      : Either 'closest2median' (default) or 'median'.
                  > 'closest2median' : The median is represented by the
                           model(s) that are closest to the
                           actual median.
                  > 'median'       : The median represented by the
                           numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc'
           is nonempty.

- 'parallel'    : Give true if you want to do the forecast in parallel.
                   Default is false.

- 'regime'      : Select the regime to do the FEVD of. Only for Markov-
                   switching model. If empty the ergodic mean will be
                   used.

- 'foundReplic' : A struct with size 1 x replic. Each element
                   must be on the same format as obj.solution. I.e.
                   each element must be the solution of the model
                   given a set of drawn parameters.

- 'stabilityTest' : Give true if you want to discard the draws
                     that give rise to an unstable model. Default
                     is false.

- 'packages'     : Cell matrix with groups of shocks and
                   the names of the groups. If you have not
                   listed a shock it will be grouped in a
                   group called 'Rest'

Must be given in the following format:

{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}

Where 'shock_name_x' must be a string with
a shock name or a cellstr array with the
shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

```

Output:

- decomp : A structure of nb_ts objects with the variance
 decomosition for each model. (If simulated it will store
 the median or closest to median dependent on the 'output'
 option)
- decompBands : A structure of structs with the percentiles
 of the variance decomosition for each model. For each
 percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the
 graph method or the nb_graphPagesGUI class to produce
 the graphs.

- `plotterBands`: A $1 \times nModel$ struct. Each field is a $1 \times nVars$ vector of `nb_graph_cs` objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_model_generic.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

■ **nb_model_group**

Go to: [Properties](#) | [Methods](#)

A class for combining models or model groups for forecast, irfs and so on.

Constructor:

```
obj = nb_model_group(models)
```

Input:

- `models` : Either a vector of `nb_model_generic` or `nb_model_group` objects or a cell array with a combination of them.

Output:

- `obj` : A `nb_model_group` object

Examples:

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- `addAutoName`
- `addAutoNameIfLocal`
- `addID`
- `addIDIfLocal`
- `forecastOutput`
- `models`
- `name`
- `valid`

- **addAutoName ↑**

Add automatic name (first) to default name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **addAutoNameIfLocal ↑**

Add automatic name (first) to local name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **models** ↑

A cell array of nb_model_generic and nb_model_group objects

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **valid** ↑

A logical array of same dimension as models. If an element is false that means that a error occured during estimation or forecasting for the corresponding model/model group.

Methods:

- aggregateForecast
- checkModel
- combineForecast
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- estimate
- evalFcstAtDates
- evaluateForecast
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getActual
- getDependentNames
- getEstimationStartDate
- getForecast
- getForecastVariables
- getHistory
- getModelNames
- getPIT
- getRecursiveScore
- getResidualNames
- getScore
- irf
- isDensityForecast
- isforecasted
- loadModelsFromPath
- mincerZarnowitzTest
- plotForecast
- plotForecastDensity
- removeObservationsOfGroup
- set
- simulateFromDensity
- storeModelsToPath
- struct
- testForecastRestrictions
- uncondForecast
- unstruct
- update

► **aggregateForecast** ↑

```
obj = aggregateForecast(obj,varargin)
```

Description:

Aggregate forecast of subcomponents of a series.

Caution: The model with the shortest forecast horizon will set the forecast horizon of the aggregated series.

Caution: A intersection of the dates from which the recursive forecast is made will be done.

Caution: If type is set to 'density' consistent paths are sampled from the marginal using a copula with an empirically estimated autocorrelation matrix.

Input:

- obj : An object of class nb_model_group

Optional Inputs:

- 'startDate' : Start date of the aggregation of forecast. I.e. the start date of which the weights are constructed. Default is to use the first shared forecast date. Must be given as a string or a nb_date object.
- 'endDate' : End date of the aggregation of forecast. I.e. the end date of which the weights are constructed. Default is to use the last shared forecast date. Must be given as a string or a nb_date object.
- 'nSteps' : The number of steps ahead forecast. The default is to take the minimum over the selected models.
- 'density' : Give 1 (true) if you want to evaluate and produce density forecast. Default is not (0 or false).

Caution: If the 'method' option is set to 'copula' the selected variables must be stationary.
- 'method' : Either:
 - > 'copula' : Use a copula to draw consistent draws from each marginal distribution. Default.
 - > 'perfectcorr' : Make the assumption that the aggregated variables are perfectly correlated, and we can therefore just sum over the distribution using the chosen weights.
- 'condLags' : Number of lags included in the calculation of the (auto)correlation matrix used when 'density' is set to true and 'method' is set to 'copula'. I.e. the number of periods to "condition on". This option should only be used if the forecast densities to be aggregated are not condition on information up until the time of the forecast. This is not the normal case, so default is 0.
- 'nLags' : Max number autocorrelation to include in the stacked autocorrelation matrix. Default is 5. If 0, only contemporaneous correlations are used. Only an option when 'density' is set to true and 'method' is set to 'copula'. If empty the full set of autocorrelations are used.
- 'weights' : The aggregation weights. As a 1 x nModels double, or a T x nModels nb_ts object (time-varying weights). Default is equal weights.

```

- 'newVar'           : A string with the name of the aggregate series.

- 'variables'       : A cellstr with same length as obj.models. These are
                      the variables from the different models to aggregate.
                      Default is to take the first variables from the
                      forecastOutput.variables list.

- 'draws'            : Number of draws from the marginal distributions of
                      the disaggregated models to base the new aggregated
                      density forecast on. Default is 1000.

- 'fcstEval'         : See the forecast method of the nb_model_generic
                      class.

- 'perc'              : Error band percentiles. As a 1 x numErrorBand
                      double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.9.
                      Can be empty, i.e. no percentiles are reported, and
                      all simulation are stored.

- 'parallel'          : When numel(obj) > 1 you may run the process in
                      parallel. true or false.

Caution: Cannot be provided if numel(obj) == 1.

- 'cores'             : Number of cores to use when the process is ran in
                      parallel. As a positive integer.

Caution: Cannot be provided if numel(obj) == 1.

- 'saveToFile'        : Logical. Save densities and domains to files. One
                      file for each model. Default is false (do not).

- 'recursive'         : Calculate the correlation matrix recursively when
                      aggregating density forecast using the 'copula'
                      method. true or false. Default is false.

- 'rollingWindow'     : If the correlation matrix should be calculated
                      recursively with a rolling window. Default is [], i.e.
                      to use the full sample to calculate the correlation
                      matrix.

- 'waitbar'            : true or false. Default is true. Not in parallel
                      session.

- 'weightsMethod'      : Either 'actual', 'end', 'ar' or 'var'. This options
                      sets the way weights are recursively extrapolated
                      when the 'weights' is set ot an object of class
                      nb_ts. See the 'method' input to the
                      recursiveExtrapolate method of the nb_ts class for
                      more on this option.

- 'sigmaMethod'        : Either 'var' or 'correlation'. If set to 'var'
                      the autocorrelation matrix is estimated using the
                      theoretical conditional moments of the forecast
                      from a VAR. If 'correlation' the autocorrelation
                      matrix is estimated based on the emprirical data
                      only. 'correlation' is default.

```

Output:

- obj : A nb_model_group object storing the aggregated forecast.

See also:

[nb_model_group.combineForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **checkModel** ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_group.

Output:

- obj : An object of class nb_model_group.

Written by Kenneth Sæterhagen Paulsen

► **combineForecast** ↑

```
obj = combineForecast(obj,varargin)
[obj,weights,plotter] = combineForecast(obj,varargin)
```

Description:

Combine density or point forecast.

Input:

- obj : An object of class nb_model_group

Optional inputs:

- 'startDate' : Start date of the combination of forecast. I.e. the start date of which the weights are constructed. Default is to use the first shared forecast date. Must be given as a string or a nb_date object.
- 'endDate' : End date of the combination of forecast. I.e. the end date of which the weights are constructed. Default is to use the last shared forecast date. Must be given as a string or a nb_date object.
- 'nSteps' : The number of steps ahead forecast. The default is to take the minimum over the selected models.
- 'type' :
 - : A string with one of the following:
 - 'MSE' : Mean squared error
 - 'RMSE' : Root mean squared errors
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'equal' : Use equal weights
- 'draws' : Number of draws from the combine density. Default is 1000.
- 'density' :
 - : Give 1 (true) if you want to evaluate and produce density forecast. Default is not (0 or false).
 - Caution: If you set this option to false the mean forecast is used to construct the point forecast from model which has produced density forecast.
- 'perc' :
 - : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.9. Can be empty, i.e. no percentiles are reported. Only an option when 'density' is set to true.
- 'allPeriods' :
 - : Give 1 (true) if you want to construct (density) forecast for all recursive forecast (which may have been produced by the models stored in this object). Default is 0 (false), i.e. only to produce forecast for the last period.
- 'optimizeOpt' :
 - : Give 1 (true), if you want to use optimized weights, instead of weights based on the formula;
 - w(i) = score(i)/sum(score)
 - Caution : Not yet finished!!!
- 'fcstEval' :
 - : See the forecast method of the nb_model_generic class.
- 'varOfInterest' :
 - : As a string or a cellstr. Use this option if you want to combine forecast for one or more variables.
- 'check' :
 - : Give true (1) to check that the domain is the same for all models. If they are not we need to simulate "new" densities at a shared domain (using 1000 bins). This could take a lot of time! To prevent this

use the 'bins' option of the forecast method of the nb_model_generic. Default is false (The error may be hard to interpret if the domain is conflicting). This may also lead to more severe density estimation errors!

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'rollingWindow' : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- 'parallel' : Set to true to run forecast combination in parallel.
- 'lambda' : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- obj : An object of class nb_model_group where the combined forecast is saved in the property forecastOutput
- weights : A struct. Where each field store the weights at forecast horizon h. Each field consist of a nDates x nModel x nVar_nb_ts object.
- plotter : A 1 x h nb_graph_ts object, which you can call the graph method on to produce a graph of the weights over the recursive periods if allPeriods is set to true.

Written by Kenneth Sætherhagen Paulsen

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_group.forecast](#), [nb_model_group.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the `nb_distribution.perc2ParamDist` function.

Input:

- `obj` : A `nb_model_forecast` object.
- `distribution` : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- `draws` : Number of draws to make from the estimated distributions.

Optional input:

- `'optimizer'` : See doc of the optimizer input to the `nb_callOptimizer` function. Default is '`fmincon`'.
- `'optimset'` : See doc of the `opt` input to the `nb_callOptimizer` function.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

[nb_model_group.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

```
obj = doSimulateFromDensity(obj,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_model_group.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments ↑**

```
[m,c]          = empiricalMoments(obj,varargin)
[m,c,ac1]      = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]  = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_group object. The options.data property of the models property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_group
- Optional inputs;
- 'vars' : A cellstr with the wanted variables.
 - 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

```
- 'nLags'      : Number of lags to compute when 'stacked' is set to true.  
- 'type'       : Either 'covariance' or 'correlation'. Default is 'correlation'.  
- 'startDate'  : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.  
- 'endDate'    : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.  
- 'demean'     : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.
```

Output:

```
- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
```

Written by Kenneth Sæterhagen Paulsen

► **estimate** ↑

```
obj = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_group object(s).

Input:

```
- obj : A vector of nb_model_group objects.
```

Optional input:

```
- varargin : See the the estimate method of the nb_model_generic class.
```

Output:

```
- obj : A vector of nb_model_group objects, where the estimation results are stored in the property results.
```

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

► evalFcstAtDates ↑

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)  
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

obj = evaluateForecast(obj,varargin)

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **forecast** ↑

obj = forecast(obj,nSteps,varargin)

Description:

Produced conditional point and density forecast of nb_model_group objects. Use the combineForecast method to produce the combined density or point forecast.

Input:

- obj : A vector of nb_model_group objects.
- nSteps : See the forecast method of the nb_model_generic class

Optional inputs:

- See the forecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_group objects. See the property forecastOutput of each model stored in the models property of the nb_model_group object.

Written by Kenneth Sæterhagen Paulsen

► forecastPerc2Dist ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_group.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_group.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_group.getActual(models,vars,startFcst,nSteps,inputs)
```

Description:

Get actual data to compare against the forecast.

Input:

- models : See the property models of the nb_model_group class.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_group objects

Input:

- obj : A vector of nb_model_group objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the model group. The return value is the estimation start date of the model that is ordered last.

Input:

- obj : An object of class nb_model_group

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

► getForecast ↑

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast output on the nb_ts format object.

Input:

- obj : An object of class nb_model_group. Must have produced forecast.
- outputType :
 - > 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.
 - The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.
 - > 'graph' : This option can be used to make it easier to plot the recursive densities.
 - The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.
 - The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are.

calculated, or else it is a empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the
wanted forecast. E.g. '2012Q1'. The fcstData will
be a nb_ts object with size nHor x nVar x nPerc + 1,
while fcstPercData will be empty. nPerc will then be
the number of percentiles/simualtions. Mean will be at
last page.
```

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' outputType. Default is false (0).

Optional inputs:

```
> 'horizon' : The fcstData output will be a nb_ts object with
size nRec x nVars. While the fcstPercData output
will be a nb_ts object with size nRec x nVars x nSim.
You can set the horizon to return by the optional
input 'horizon'. See the 'timing' option also.
```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_model_group.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

► **getForecastVariables ↑**

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_group object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual vary with the date of recursion, so by this option you can get the residual of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth SÃ¶terhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getPIT ↑**

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidualNames ↑**

residualNames = getResidualNames(obj)

Description:

Get all the unique residual names of a vector of nb_model_group object

Input:

- obj : A vector of nb_model_group objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...)
                    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when

constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **irf ↑**

[irfs,irfsBands,plotter] = irf(obj,varargin)

Description:

Produce IRFs of the model(s) represented by nb_model_group object.

Input:

- obj : A nb_model_group object.

Optional input:

- varargin : See the the irf method of the nb_model_generic class.

Output:

- [irfs,irfsBands,plotter] : See the the irf method of the nb_model_generic class.

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **loadModelsFromPath** ↑

```
obj = nb_model_group.loadModelsFromPath(pathName)
```

Description:

Load models saved to separate .mat files stored in a folder to a model group object. The models must be stored as nb_model_generic or nb_model_group objects.

Input:

```
- pathName : Full path name of the folder the model files are stored.  
As a string.
```

Output:

```
- obj : An object of class nb_model_group
```

Written by Kenneth SÃ¶terhagen Paulsen

► **mincerZarnowitzTest ↑**

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

Input:

```
- obj : An object of class nb_model_group. You need to call the  
combineForecast method first!  
  
- precision : The precision of the printed result. As a string. Default  
is '%8.6f'.
```

Output:

```
- test : A nb_ts object with the test statistic. As a  
nHor x nModel x nVar nb_ts object.  
  
- pval : A nb_ts object with the p-values of the test. As a  
nHor x nModel x nVar nb_ts object.  
  
- res : A char with the printout of the test.
```

See also:

[nb_model_generic.combineForecast](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **plotForecast ↑**

```
plotter = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► removeObservationsOfGroup ↑

obj = removeObservationsOfGroup(obj, numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_model_group or nb_model_selection_group object.
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_model_group or nb_model_selection_group object.

See also:

`nb_model_selection_group.removeObservations`

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

`obj = set(obj,varargin)`

Description:

Sets the properties of the nb_model_group objects.

Input:

- `obj` : A vector of nb_model_group objects.

Optional input:

If number of inputs equals 1:

- `varargin{1}` : A structure of fields to be set.

Else:

- `varargin` : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object.

Output:

- `obj` : A vector of nb_model_group objects.

Written by Kenneth Sæterhagen Paulsen

► **simulateFromDensity** ↑

`obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `forecastOutput` : A forecast property of the nb_model_forecast class where the `forecastOutput.evaluation` stores the kernel density estimate.

- `draws` : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **storeModelsToPath** ↑

storeModelsToPath(obj, pathName)

Description:

Save the different models of the nb_model_group object to seperate .mat files to a given folder.

Input:

- obj : An object of class nb_model_group
- pathName : Full path name of the folder the model files should be stored. As a string. Must be ended with a '\'.

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_group.

Output:

- s : A struct representing the nb_model_group object.

See also:

nb_model_group.unstruct

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_group objects. Use the combineForecast method to produce the combined density or point forecast.

Input:

- obj : A vector of nb_model_group objects.
- nSteps : See the uncondForecast method of the nb_model_generic class

Optional inputs:

- See the uncondForecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_group objects. See the property forecastOutput of each model stored in the models property of the nb_model_group object.

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

obj = nb_model_group.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_group class.

Input:

- s : A struct. See nb_model_group.struct.

Output:

- obj : An object which is a subclass of the nb_model_group class.

See also:

[nb_model_group.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model group, and optionally re-estimate, re-solve and forecast the models given the updated data

Input:

- obj : A vector of objects of class nb_model_group
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.
- groupIndex : A 1 x nLevel double with the group level.

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if estimation or forecasting failes. The models that failes will then be removed when calling methods as compareForecast and aggregateForecast etc. See the valid property.

Output:

- obj : A vector of objects of class nb_model_group.
- valid : A logical with size 1 x nObj. true at location ii if updating of model group ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

■ nb_model_recursive_detrending

Go to: [Properties](#) | [Methods](#)

Do recursive de-trending of a model and produce recursive forecast based on this recursive detrending. The resulting object can be used by nb_model_group as a normal nb_model_generic object.

Constructor:

```
- obj = nb_model_recursive_detrending(model,varargin)

Input:

- model : An object of a subclass of the nb_model_generic class.

Optional input:

- See the set method for more on the inputs.
  (nb_model_recursive_detrending.set)

Output:

- obj : An object of class nb_model_recursive_detrending.
```

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [addAutoName](#)
- [addAutoNameIfLocal](#)
- [addID](#)
- [addIDIfLocal](#)
- [forecastOutput](#)
- [model](#)
- [name](#)
- [options](#)
- [reporting](#)
- [results](#)
- [solution](#)
- [transformations](#)
- [userData](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[forecastOutput](#)** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **model** ↑

The underlying model to use for forecasting. As a subclass of the nb_model_generic class.

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that persevere the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **results** ↑

A struct storing all the results of the estimation.

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F + G*P_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

```
X_t = G*Z_t + Y_t
```

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors (exogenous variable in the case of ARIMA models).
- F : See the equation above. A nobs x 1 double.
(Storing the constant terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

- **transformations** [↑](#)

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; `methods('nb_math_ts')`.

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** [↑](#)

Adds the possibility to add user data. Can be of any type

Methods:

- appendData
- checkModel
- checkReporting
- convert
- convertEach
- createVariables
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- estimate
- evalFcstAtDates
- evaluateForecast
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getDependent
- getDependentNames
- getEstimationStartDate
- getFiltered
- getForecast
- getForecastVariables
- getHistory
- getModelNames
- getOriginalVariables
- getPIT
- getPosteriorDistributions
- getPredicted
- getRecursiveEstimationGraph
- getRecursivePeriods
- getRecursiveScore
- getResidual
- getResidualGraph
- getResidualNames
- getRoots
- getScore
- help
- isDensityForecast
- isFiltered
- isMS
- isbayesian
- isestimated
- isforecasted
- issolved
- plotForecast
- plotForecastDensity
- print
- print_estimation_results
- removeObservations
- set
- simulateFromDensity
- solve
- struct
- template
- testForecastRestrictions
- uncondForecast
- unstruct
- update

► **appendData ↑**

```
obj = appendData(obj, DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► checkModel ↑

obj = checkModel(obj)

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_recursive_detrending.

Output:

- obj : An object of class nb_model_recursive_detrending.

Written by Kenneth Sæterhagen Paulsen

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables

Input:

- obj : A scalar nb_model_recursive_detrending object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

► **convert** ↑

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A 1x1 nb_model_recursive_detrending object
- expressions : A N x 4 x nPeriods cell matrix of how to transform input data to model variables recursively. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
- > third column : Any expression that can be interpreted by the nb_ts.createShift method.
- > fourth column : Comments

The third dimension of this input refers to the number of recursive period you want to estimate or forecast your model. This allow for time-varying detrending, as for example real-time hp-filtering.

Caution : The size of the third dimension must be equal to recursive period given by the recursive_start_date and recursive_end_date options.

- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A 1 x nPeriods vector of nb_graph_ts object with a graph with the detrending of each recursive period. Use the graphInfoStruct method to produce the graph of each object. Do not call it on the hole vector!!

Examples:

```
expressions = {%
    Name,      input,      shift,      description
    'QSA_DPQ_YMN',   'log(QSA_YMN)',   {'hpfilter',1600}, '',
    'QSA_DPQ_PCPIJAE', 'log(QSA_DPQ_PCPIJAE)', {'hpfilter',1600}, '',
    'QSA_DPQ_CP',     'log(QSA_DPQ_CP)',   {'hpfilter',1600}, ''};

modelRec = modelRec.createVariables(expressions)
```

See also:

[nb_model_recursive_detrending.reporting](#)

► **doForecastPerc2Dist** ↑

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_recursive_detrending.forecast](#), [nb_model_recursive_detrending.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_recursive_detrending.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity ↑**

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_model_recursive_detrending.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments ↑**

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_group object. The options.data property of the model property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_recursive_detrending.

Optional inputs;

- 'vars' : A cellstr with the wanted variables.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

► **estimate ↑**

```
obj = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_recursive_detrending object(s).

Caution: If the underlying model is of class nb_dsg, only recursive filtering is done with the call to this function!

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Optional input:

- Optional inputs given to the nb_dsg.filter method, if the model property is of class nb_dsg. Otherwise it has no effect.

Caution : The 'endDate' inputs to the filter method will be overwritten!

Output:

- obj : A vector of nb_model_recursive_detrending objects, where the estimation results are stored in the property results.

Written by Kenneth Sæterhagen Paulsen

► evalFcstAtDates ↑

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:

```

- 'RMSE'   : Root mean squared error
- 'MSE'    : Mean squared error
- 'MAE'    : Mean absolute error
- 'MEAN'   : Mean error
- 'STD'    : Standard error of the forecast error
- 'ESLS'   : Exponential of the sum of the log scores
- 'EELS'   : Exponential of the mean of the log scores
- 'MLS'    : Mean log score

- density : Set true if you have a density forecast.

- quart   : Set true if you want to compare the quarterly forecasts. This
            option automatically detects and uses the quarterly estimates
            when evaluating.

```

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **forecast** ↑

obj = forecast(obj,nSteps,varargin)

Description:

Produced recursive conditional point and density forecast of nb_model_recursive_detrending objects.

Input:

- obj : A vector of nb_model_recursive_detrending objects.
- nSteps : See the forecast method of the nb_model_generic class

Optional inputs:

- See the forecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_recursive_detrending objects. See the property forecastOutput.

Written by Kenneth Sæterhagen Paulsen

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_recursive_detrending.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist ↑**

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_recursive_detrending.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åstrøm Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_recursive_detrending object.

Caution: Only returned for the last period of the recursive estimation.

Input:

- obj : An object of class nb_model_recursive_detrending

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Åstrøm Paulsen

► **getDependentNames ↑**

dependentNames = getDependentNames(obj)

Description:

Get all the unique dependent variables names of a vector of nb_model_recursive_detrending objects

Input:

- obj : A vector of nb_model_recursive_detrending objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

► getEstimationStartDate ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_recursive_detrending

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_recursive_detrending
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.

- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Output:

- data : A nb_ts object with size nobs x nvars x nPeriods included the filtered variables.

See also:

[nb_model_recursive_detrending.estimate](#)

Written by Kenneth Åkerhagen Paulsen

► **getForecast ↑**

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.
- outputType :
 - > 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.
 - The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.
- > 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date'      : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

> 'horizon'   : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).
```

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_model_recursive_detrending.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables** ↑

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

histData = getHistory(obj,vars,date)

Description:

Get historical data

Input:

- obj : A scalar nb_model_recursive_detrending object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables is found.

- date : For recursively detrended and estimated models, the residual and data vary with the date of recursion, so by this option you can get the residual of a given recursion.

The same apply for real-time data.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)

- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

```
- pit      : A nb_data object with size nPeriods x nHor x nVars  
- plotter : A nb_graph_data object. please use the graph method, if you  
want to plot it.  
- pitTest : Test for a uniformly distributed PITs using a Pearson  
chi-squared test. As a 2 x nHor x nVar nb_cs object.
```

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getPosteriorDistributions** ↑

```
[distr,paramNames] = getPosteriorDistributions(obj)
```

Description:

Get posterior distributions of model.

Input:

```
- obj      : An object of class nb_model_recursive_detrending.
```

Output:

```
- distr      : A nPeriods x numCoeff nb_distribution object.  
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.
```

Written by Kenneth Sæterhagen Paulsen

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_recursive_detrending object

Caution: Only returned for the last period of the recursive estimation.

Input:

- obj : An object of class nb_model_recursive_detrending

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

► **getRecursiveEstimationGraph ↑**

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_recursive_detrending.

Output:

- plotter : An object of class nb_graph.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

► **getRecursivePeriods ↑**

```
periods          = getRecursivePeriods(obj)
[periods,dates,obj] = getRecursivePeriods(obj)
```

Description:

Get the number of recursive periods to be looped.

Input:

- obj : An object of class nb_model_recursive_detrending.

Output:

- periods : The number of recursive periods, as a scalar double.
- obj : An object of class nb_model_recursive_detrending where the options property is updated with the corrected fields 'recursive_start_date'.

Written by Kenneth Sæterhagen Paulsen

► getRecursiveScore ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_recursive_detrending object

Input:

- obj : An object of class nb_model_recursive_detrending

Output:

- residual : Either a nb_ts object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_recursive_detrending.

Output:

- plotter : An object of class nb_graph.

Examples:

```
plotter = getResidualGraph(obj);
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sætherhagen Paulsen

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sætherhagen Paulsen

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert, ...
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object intput is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **help ↑**

```
helpText = nb_model_recursive_detrending.help
helpText = nb_model_recursive_detrending.help(option)
helpText = nb_model_recursive_detrending.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► isDensityForecast ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► isFiltered ↑

```
ret = isFiltered(obj)
ret = isFiltered(obj,tested)
```

Description:

Test if a nb_model_recursive_detrending object is filtered or not.

Input:

- obj : An object of class nb_model_recursive_detrending.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_recursive_detrending object a Markov switching model or not.

Input:

- obj : A nb_model_recursive_detrending object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_recursive_detrending object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_recursive_detrending object empty (not estimated) or not.

Input:

- obj : A nb_model_recursive_detrending object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sætherhagen Paulsen

► plotForecast ↑

```
plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.

- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_recursive_detrending.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_recursive_detrending.print](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations ↑**

obj = removeObservations(obj,numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties of the nb_model_recursive_detrending objects.

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object.

Output:

- obj : A vector of nb_model_recursive_detrending objects.

Written by Kenneth Sæterhagen Paulsen

► **simulateFromDensity ↑**

obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **solve** ↑

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_model_recursive_detrending object(s).

Input:

- obj : A vector of nb_model_recursive_detrending objects.

Output:

- obj : A vector of nb_model_recursive_detrending objects, where the solved model(s) is/are stored in the property solution.

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_recursive_detrending.

Output:

- s : A struct representing the nb_model_recursive_detrending object.

See also:

[nb_model_group.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

► **template** ↑

```
options = nb_model_recursive_detrending.template()
options = nb_model_recursive_detrending.template(num)
```

Description:

Construct a struct which can be provided to the nb_model_recursive_detrending class constructor.

This structure provided the user the possibility to set different options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

See also:

[nb_model_recursive_detrending](#)

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_C_P'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **uncondForecast ↑**

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_group objects. Use the combineForecast method to produce the combined density or point forecast.

Input:

- obj : A vector of nb_model_recursive_detrending objects.
- nSteps : See the uncondForecast method of the nb_model_generic class

Optional inputs:

- See the uncondForecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_recursive_detrending objects. See the property forecastOutput.

Written by Kenneth Sæterhagen Paulsen

► **unstruct** ↑

```
obj = nb_model_recursive_detrending.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_recursive_detrending class.

Input:

- s : A struct. See nb_model_recursive_detrending.struct.

Output:

- obj : An object of the nb_model_recursive_detrending class.

See also:

[nb_model_recursive_detrending.struct](#)

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

```
obj = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model group, and optionally re-estimate, re-solve and forecast the models given the updated data

Input:

- obj : A vector of objects of class nb_model_recursive_detrending
- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve
> 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'.
- groupIndex : A 1 x nLevel double with the group level.

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if estimation or forecasting failes. The models that failes will then be removed when calling methods as compareForecast and aggregateForecast etc. See the valid property.

Output:

- obj : A vector of objects of class nb_model_recursive_detrending.
- valid : A logical with size 1 x nObj. true at location ii if updating of model group ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

■ nb_model_selection_group

Go to: [Properties](#) | [Methods](#)

A class for doing model selection.

Superclasses:

nb_model_group, nb_modelData

Constructor:

```
obj = nb_model_selection_group(models)
```

Input:

- models : A 1 x N vector of nb_model_generic object.

Output:

- obj : An object of class nb_model_selection_group

Examples:

See also:

[nb_model_group](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- addAutoName
- addAutoNameIfLocal
- addID
- addIDIfLocal
- forecastOutput
- models
- name
- options
- reporting
- transformations
- valid

- **addAutoName** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addAutoNameIfLocal** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addID** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **addIDIfLocal** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **models** ↑

A cell array of nb_model_generic and nb_model_group objects

Inherited from superclass NB_MODEL_GROUP

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **valid** ↑

A logical array of same dimension as models. If an element is false that means that an error occurred during estimation or forecasting for the corresponding model/model group.

Inherited from superclass NB_MODEL_GROUP

Methods:

- aggregateForecast
- appendData
- checkModel
- checkReporting
- combineForecast
- convert
- convertEach
- createVariables
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- estimate
- evalFcstAtDates
- evaluateForecast
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getActual
- getDependentNames
- getEstimationStartDate
- getForecast
- getForecastVariables
- getHistory
- getModelNames
- getOriginalVariables
- getPIT
- getRecursiveScore
- getResidualNames
- getScore
- help
- irf
- isDensityForecast
- isforecasted
- loadModelsFromPath
- mincerZarnowitzTest
- modelSelection
- plotForecast
- plotForecastDensity
- removeObservations
- removeObservationsOfGroup
- set
- simulateFromDensity
- storeModelsToPath
- struct
- template
- testForecastRestrictions
- uncondForecast
- unstruct
- update

► **aggregateForecast ↑**

```
obj = aggregateForecast(obj,varargin)
```

Description:

Aggregate forecast of subcomponents of a series.

Caution: The model with the shortest forecast horizon will set the forecast horizon of the aggregated series.

Caution: A intersection of the dates from which the recursive forecast is made will be done.

Caution: If type is set to 'density' consistent paths are sampled from the marginal using a copula with an empirically estimated autocorrelation matrix.

Input:

- obj : An object of class nb_model_group

Optional Inputs:

- 'startDate' : Start date of the aggregation of forecast. I.e. the start date of which the weights are constructed. Default is to use the first shared forecast date. Must be given as a string or a nb_date object.

- 'endDate' : End date of the aggregation of forecast. I.e. the end date of which the weights are constructed. Default is to use the last shared forecast date. Must be given as a string or a nb_date object.

- 'nSteps' : The number of steps ahead forecast. The default is to take the minimum over the selected models.

- 'density' : Give 1 (true) if you want to evaluate and produce density forecast. Default is not (0 or false).

Caution: If the 'method' option is set to 'copula' the selected variables must be stationary.

Caution: If you set this option to false the mean forecast is used to construct the point forecast from model which has produced density forecast.

- 'method' : Either:

> 'copula' : Use a copula to draw consistent draws from each marginal distribution. Default.

> 'perfectcorr' : Make the assumption that the aggregated variables are perfectly correlated, and we can therefore just sum over the distribution using the chosen weights.

- 'condLags' : Number of lags included in the calcualtion of the (auto)correlation matrix used when 'density' is set to true and 'method' is set to 'copula'. I.e. the

number of periods to "condition on". This option should only be used if the forecast densities to be aggregated are not condition on information up until the time of the forecast. This is not the normal case, so default is 0.

- 'nLags' : Max number autocorrelation to include in the stacked autocorrelation matrix. Default is 5. If 0, only contemporaneous correlations are used. Only an option when 'density' is set to true and 'method' is set to 'copula'. If empty the full set of autocorrelations are used.
- 'weights' : The aggregation weights. As a 1 x nModels double, or a T x nModels nb_ts object (time-varying weights). Default is equal weights.
- 'newVar' : A string with the name of the aggregate series.
- 'variables' : A cellstr with same length as obj.models. These are the variables from the different models to aggregate. Default is to take the first variables from the forecastOutput.variables list.
- 'draws' : Number of draws from the marginal distributions of the disaggregated models to base the new aggregated density forecast on. Default is 1000.
- 'fcstEval' : See the forecast method of the nb_model_generic class.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.9. Can be empty, i.e. no percentiles are reported, and all simulation are stored.
- 'parallel' : When numel(obj) > 1 you may run the process in parallel. true or false.
 Caution: Cannot be provided if numel(obj) == 1.
- 'cores' : Number of cores to use when the process is ran in parallel. As a positive integer.
 Caution: Cannot be provided if numel(obj) == 1.
- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'recursive' : Calculate the correlation matrix recursively when aggregating density forecast using the 'copula' method. true or false. Default is false.
- 'rollingWindow' : If the correlation matrix should be calculated recursively with a rolling window. Default is [], i.e. to use the full sample to calculate the correlation matrix.
- 'waitbar' : true or false. Default is true. Not in parallel

session.

- 'weightsMethod' : Either 'actual', 'end', 'ar' or 'var'. This option sets the way weights are recursively extrapolated when the 'weights' is set to an object of class nb_ts. See the 'method' input to the recursiveExtrapolate method of the nb_ts class for more on this option.
- 'sigmaMethod' : Either 'var' or 'correlation'. If set to 'var' the autocorrelation matrix is estimated using the theoretical conditional moments of the forecast from a VAR. If 'correlation' the autocorrelation matrix is estimated based on the empirical data only. 'correlation' is default.

Output:

- obj : A nb_model_group object storing the aggregated forecast.

See also:

[nb_model_selection_group.combineForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **appendData** ↑

obj = appendData(obj,DB)

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **checkModel** ↑

obj = checkModel(obj)

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_group.

Output:

- obj : An object of class nb_model_group.

Written by Kenneth Sæterhagen Paulsen

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **combineForecast** ↑

```
obj = combineForecast(obj,varargin)
[obj,weights,plotter] = combineForecast(obj,varargin)
```

Description:

Combine density or point forecast.

Input:

- obj : An object of class nb_model_group

Optional inputs:

- 'startDate' : Start date of the combination of forecast. I.e. the start date of which the weights are constructed. Default is to use the first shared forecast date. Must be given as a string or a nb_date object.
- 'endDate' : End date of the combination of forecast. I.e. the end date of which the weights are constructed. Default is to use the last shared forecast date. Must be given as a string or a nb_date object.
- 'nSteps' : The number of steps ahead forecast. The default is to take the minimum over the selected models.
- 'type' : A string with one of the following:
 - 'MSE' : Mean squared error
 - 'RMSE' : Root mean squared errors
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'equal' : Use equal weights
- 'draws' : Number of draws from the combine density. Default is 1000.
- 'density' : Give 1 (true) if you want to evaluate and produce density forecast. Default is not (0 or false).

Caution: If you set this option to false the mean forecast is used to construct the point forecast from model which has produced density forecast.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.9. Can be empty, i.e. no percentiles are reported. Only an option when 'density' is set to true.
- 'allPeriods' : Give 1 (true) if you want to construct (density) forecast for all recursive forecast (which may have been produced by the models stored in this object). Default is 0 (false), i.e. only to produce forecast for the last period.
- 'optimizeOpt' : Give 1 (true), if you want to use optimized weights, instead of weights based on the formula;

```

w(i) = score(i)/sum(score)

Caution : Not yet finished!!!

- 'fcstEval'      : See the forecast method of the nb_model_generic
                      class.

- 'varOfInterest' : As a string or a cellstr. Use this option if you want
                      to combine forecast for one or more variables.

- 'check'          : Give true (1) to check that the domain is the same
                      for all models. If they are not we need to simulate
                      "new" densities at a shared domain (using 1000
                      bins). This could take a lot of time! To prevent this
                      use the 'bins' option of the forecast method
                      of the nb_model_generic. Default is false (The error
                      may be hard to interpret if the domain is
                      conflicting). This may also lead to more
                      severe density estimation errors!

- 'saveToFile'    : Logical. Save densities and domains to files. One
                      file for each model. Default is false (do not).

- 'rollingWindow' : Set it to a number if the combination weights are to
                      be calculated using a rolling window. Default is
                      empty, i.e. to calculate the weights recursively using
                      the full history at each recursive step.

- 'parallel'       : Set to true to run forecast combination in parallel.

- 'lambda'         : Give the value of the parameter of the exponential
                      decaying weights on past forecast errors when
                      constructing the score. If empty the weights on all
                      past forecast errors are equal.

```

Output:

- obj : An object of class nb_model_group where the combined
 forecast is saved in the property forecastOutput
- weights : A struct. Where each field store the weights at forecast
 horizon h. Each feild consist of a nDates x nModel x nVar
 nb_ts object.
- plotter : A 1 x h nb_graph_ts object, which you can call the graph
 method on to produce a graph of the weights over the
 recursive periods if allPeriods is set to true.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **convert ↑**

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables ↑**

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = {%
    Name,      input,    shift,   description
'VAR1_G',   'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',   'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',   'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **doForecastPerc2Dist** ↑

```
obj = doForecastPerc2Dist(obj, draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_selection_group.forecast](#), [nb_model_selection_group.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

```
[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_selection_group.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity ↑**

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_model_generic.forecast`, `nb_model_selection_group.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_group` object. The `options.data` property of the `models` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- `obj` : An object of class `nb_model_group`

Optional inputs;

- `'vars'` : A cellstr with the wanted variables.
- `'output'` : Either `'nb_cs'` or `'double'`. Default is `'nb_cs'`.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when `'stacked'` is set to true.
- `'type'` : Either `'covariance'` or `'correlation'`. Default is `'correlation'`.
- `'startDate'` : The start date of the calculations. Default is the start date of the `options.data` property. Only for time-series! Must be a string or a `nb_date` object.
- `'endDate'` : The end date of the calculations. Default is the end date of the `options.data` property. Only for time-series! Must be a string or a `nb_date` object.
- `'demean'` : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **estimate** ↑

obj = estimate(obj,varargin)

Description:

Estimate the model(s) represented by nb_model_group object(s).

Input:

- obj : A vector of nb_model_group objects.

Optional input:

- varargin : See the the estimate method of the nb_model_generic class.

Output:

- obj : A vector of nb_model_group objects, where the estimation results are stored in the property results.

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **evalFcstAtDates** ↑

[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...
quart)

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► forecast ↑

obj = forecast(obj,nSteps,varargin)

Description:

Produced conditional point and density forecast of nb_model_group objects. Use the combineForecast method to produce the combined density or point forecast.

Input:

- obj : A vector of nb_model_group objects.
- nSteps : See the forecast method of the nb_model_generic class

Optional inputs:

- See the forecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_group objects. See the property forecastOutput of each model stored in the models property of the nb_model_group object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_selection_group.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_selection_group.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getActual](#) ↑

```
actual = nb_model_group.getActual(models, vars, startFcst, nSteps, inputs)
```

Description:

Get actual data to compare against the forecast.

Input:

- models : See the property models of the nb_model_group class.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► getDependentNames ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_group objects

Input:

- obj : A vector of nb_model_group objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► getEstimationStartDate ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the model group. The return value is the estimation start date of the model that is ordered last.

Input:

- obj : An object of class nb_model_group

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **getForecast** ↑

```
fcstData          = getForecast (obj)
[fcstData,fcstPercData] = getForecast (obj,outputType)
[fcstData,fcstPercData] = getForecast (obj,outputType,includeHist,varargin)
```

Description:

Get the forecast output on the nb_ts format object.

Input:

- obj : An object of class nb_model_group. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is a empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they is calculated, or else it is a empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1,
```

while fcstPercData will be empty. nPerc will then be the number of percentiles/simualtions. Mean will be at last page.

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_model_selection_group.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_group object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual vary with the date of recursion, so by this option you can get the residual of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will
get the PIT from the start date of the recursive forecast and
use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct
the PITs based on the period from the estimation start date
until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a

nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:

```

        - 'RMSE'   : Root mean squared error
        - 'MSE'    : Mean squared error
        - 'MAE'    : Mean absolute error
        - 'MEAN'   : Mean error
        - 'STD'    : Standard error of the forecast error
        - 'ESLS'   : Exponential of the sum of the log scores
        - 'EELS'   : Exponential of the mean of the log scores
        - 'MLS'    : Mean log score

- dim          : Either 'variables' to get the recursive scores of all
                  the variables forecast by a model (the obj input must
                  be a scalar object), or else the name of the variable
                  to get the score of (must be part of all models!).

- startDate    : The date to begin constructing the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- endDate      : The date to end constructiong the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- invert        : true | false. Default true. If true it will invert the
                  score for 'RMSE', 'MSE', 'MAE' and 'STD'.

```

Output:

```

- score     : A nb_ts object with size nPeriods x nVars x nHorizon if dim
               equal 'variables', else a nb_ts object with size nPeriods x
               nObj x nHorizon.

- plotter   : A nb_graph_ts object to plot the recursive scores. Use the
               graph method. One figure per horizon!

```

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidualNames ↑**

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_group object

Input:

```
- obj         : A vector of nb_model_group objects
```

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert, ...
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log scoreIf the object input is a vector, the type input can also be a cellstr array with matching length.
- Caution: See comment to the invert input!
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give

NOT inverted root mean squared error!

- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **help ↑**

```
helpText = nb_model_selection_group.help
helpText = nb_model_selection_group.help(option)
helpText = nb_model_selection_group.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Produce IRFs of the model(s) represented by nb_model_group object.

Input:

- obj : A nb_model_group object.

Optional input:

- varargin : See the the irf method of the nb_model_generic class.

Output:

- [irfs,irfsBands,plotter] : See the the irf method of the nb_model_generic class.

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **loadModelsFromPath** ↑

```
obj = nb_model_group.loadModelsFromPath(pathName)
```

Description:

Load models saved to separate .mat files stored in a folder to a model group object. The models must be stored as nb_model_generic or nb_model_group objects.

Input:

- pathName : Full path name of the folder the model files are stored.
As a string.

Output:

- obj : An object of class nb_model_group

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **mincerZarnowitzTest** ↑

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

Input:

- obj : An object of class nb_model_group. You need to call the combineForecast method first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_model_generic.combineForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► [modelSelection ↑](#)

```
modelGroup = modelSelection(obj);
[modelGroup,plotter] = modelSelection(obj);
[modelGroup,plotter,errorReport] = modelSelection(obj)
```

Description:

Select models based on their forecasting properties.

- 'class' == 'nb_var':

This function will test combinations of the variables in the data and with different lag length.

This code is inspired by a script made by Martin Blomhoff Holm,
Norges Bank 03/06/2014

- 'class' == 'nb_arima'

This function will test combinations of AR and MA terms.

Caution: This code will not produce the final combined forecast,
use the combineForecast method of nb_model_group class on the output to produce these forecast.

Output:

- modelGroup : A cell array with size 1xM, with the best models at different horizons. See the 'nHor' option.
Caution : If the 'setUpOnly' option is set to true this output is a 1 x num vector of nb_var models.
- plotter : A graph showing the most selected variables in the final selection of models. Shown in a histogram. Use the graph method on the returned object to plot it on the screen. An object of class nb_graph_cs.

See also:

[nb_model_group.combineForecast](#), [nb_getCombinations](#)

Written by Kenneth Sæterhagen Paulsen

► [plotForecast ↑](#)

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **removeObservationsOfGroup** ↑

```
obj = removeObservationsOfGroup(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_model_group or nb_model_selection_group object.
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_model_group or nb_model_selection_group object.

See also:

`nb_model_selection_group.removeObservations`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **set ↑**

`obj = set(obj,varargin)`

Description:

Sets the properties of the nb_model_selection_group objects.

Input:

- `obj` : A vector of nb_model_selection_group objects.

Optional input:

If number of inputs equals 1:

- `varargin{1}` : A structure of fields to be set.

Else:

- `varargin` : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object.

Output:

- `obj` : A vector of nb_model_selection_group objects.

Written by Henrik Halvorsen Hortemo

► **simulateFromDensity ↑**

`obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)`

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **storeModelsToPath ↑**

storeModelsToPath(obj, pathName)

Description:

Save the different models of the nb_model_group object to seperate .mat files to a given folder.

Input:

- obj : An object of class nb_model_group
- pathName : Full path name of the folder the model files should be stored. As a string. Must be ended with a '\'.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **struct ↑**

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_group.

Output:

- s : A struct representing the nb_model_group object.

See also:

[nb_model_selection_group.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **template** ↑

```
options = nb_model_selection_group.template(num)
```

Description:

Construct a struct which can be provided to the nb_model_selection_group class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

See also:

[nb_model_generic](#)

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

```

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
    {variable, date, test}.

    > variable : The variable(s) to test,
        as a string or cell array of strings.

    > date : The date as a string. If a cell array is given, a
        copy of the restriction will be made for every given
        date.

    > test : Restriction test as a function handle. The value(s)
        of the variable(s) in 'variable' at time 'date' are
        passed as arguments. Should return a logical.

```

Output:

- out : A double with the probability

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
    {'QSA_URR', 'QSA_DPO_C_P'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);

```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **uncondForecast ↑**

obj = uncondForecast(obj,nSteps,varargin)

Description:

Produced unconditional point and density forecast of nb_model_group objects. Use the combineForecast method to produce the combined density or point forecast.

Input:

```

- obj : A vector of nb_model_group objects.

- nSteps : See the uncondForecast method of the nb_model_generic class

```

Optional inputs:

- See the uncondForecast method of the nb_model_generic class

Output:

- obj : A vector of nb_model_group objects. See the property forecastOutput of each model stored in the models property of the nb_model_group object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **unstruct** ↑

obj = nb_model_group.unstruct(s)

Description:

Convert a struct to an object which is a subclass of the nb_model_group class.

Input:

- s : A struct. See nb_model_group.struct.

Output:

- obj : An object which is a subclass of the nb_model_group class.

See also:

[nb_model_selection_group.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

► **update** ↑

obj = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)

Description:

Update data of model group, and optionally re-estimate, re-solve and forecast the models given the updated data

Input:

- obj : A vector of objects of class nb_model_group
- type : > '' : Only update data. Default
> 'estimate' : Update data and estimate.
> 'solve' : Update data, estimate and solve

```

        > 'forecast' : Update data, estimate, solve and forecast.

- warningOn    : 'on' or 'off'. If 'on' only warnings are given while
                  the data source is updated, else an error will be given.
                  Default is 'on'. Set it to 'off', if the 'write' option
                  is used.

- inGUI        : 'on' or 'off'. Indicate if the update command is called
                  in the GUI or not. Default is 'off'.

- groupIndex   : A 1 x nLevel double with the group level.

```

Optional input:

```

- 'write'      : Give this string as one of the optional inputs to write
                  an error file if estimation or forecasting failes. The
                  models that failes will then be removed when calling
                  methods as compareForecast and aggregateForecast etc.
                  See the valid property.

```

Output:

```

- obj       : A vector of objects of class nb_model_group.

- valid    : A logical with size 1 x nObj. true at location ii if
              updating of model group ii succeeded, otherwise false. If
              'write' is not used an error will be thrown instead, so in
              this case this output will be true for all models.

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GROUP

■ nb_nonLinearEq

Go to: [Properties](#) | [Methods](#)

A class for estimation of single non-linear equation models.

Superclasses:

nb_model_generic

Constructor:

```
obj = nb_nonLinearEq(varargin)
```

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_nonLinearEq object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#), [nb_nonLinearEq.template](#)
[nb_nonLinearEq.help](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [addAutoName](#)
- [dependent](#)
- [forecastOutput](#)
- [parser](#)
- [solution](#)
- [addAutoNameIfLocal](#)
- [endogenous](#)
- [name](#)
- [reporting](#)
- [transformations](#)
- [addID](#)
- [estOptions](#)
- [options](#)
- [residuals](#)
- [userData](#)
- [addIDIfLocal](#)
- [exogenous](#)
- [parameters](#)
- [results](#)

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgen models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **[endogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsgen models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **parser** ↑

This property will store the info on the model, such as equations etc, otherwise it will be empty.

Caution: If you are dealing with an object of class nb_dsg, this property will be empty, if the DSGE model is parsed with RISE or Dynare.

Inherited from superclass NB_MODEL_PARSE

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

```
O_t = F*X_t + G*Y_t + u_t, u ~ N(0,R)
```

Observation equation (ARIMA models only):

```
X_t = G*Z_t + Y_t
```

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.
As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres)

indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- addConstraints
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFcstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constraints
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph

- `getResidualNames`
- `getScore`
- `getVariablesList`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `parse`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`
- `removeObservations`
- `shock_decomposition`
- `simulateFromDensity`
- `solveVector`
- `template`
- `testParameters`
- `uncertaintyDecomposition`
- `unstruct`
- `variance_decomposition`
- `getRoots`
- `getSolution`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `irf`
- `isFiltered`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `parseTempParameters`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `set`
- `simulate`
- `simulatedMoments`
- `struct`
- `testForecastRestrictions`
- `theoreticalMoments`
- `uncondForecast`
- `update`

► **`addConstraints`** ↑

```
obj = addConstraints(obj,constraints)
```

Description:

Add (more) constraints to the parameters during estimation. The constraints can only include parameters (and numbers).

Input:

- obj : An object of class nb_dsg or nb_nonLinearEq.
- constraints : A N x M char array or a N x 1 (or 1 x N) cellstr array.
Each element will be counted as a new constraint.

Output:

- obj : An object of class nb_dsg or nb_nonLinearEq.

See also:

[nb_dsg.parse](#), [nb_nonLinearEq.parse](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

obj = assignParameters(obj,varargin)

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- `obj` : An object of class `nb_model_generic`, where the parameters of the underlying model has been changed. See `solve` to get the solution of the model with the updated parameters.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **assignPosteriorDraws** ↑

`obj = assignPosteriorDraws(obj,varargin)`

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- `'param'` : The names of the assign parameters. As a `cellstr` vector.
- `'value'` : The values of the assign parameters. As a double `nParam x nReg x nDraws` vector.

> Optional number of input pairs:

- `'beta'` : Assign all the posterior draws of the parameters of the main equation. Must be of same size as `obj.results.beta`. The draws must come in the third dimension.
- `'sigma'` : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as `obj.results.sigma`. The draws must come in the third dimension.
- `'lambda'` : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as `obj.results.lambda`. Only for factor models. The draws must come in the third dimension.
- `'R'` : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as `obj.results.R`. Only for factor models. The draws must come in the third dimension.
- `'period'` : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► assignTexNames ↑

```
obj = assignTexNames(obj, names, texNames)
```

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as $_t$ is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► calculateMultipliers ↑

```
mult = calculateMultipliers(obj, varargin)
```

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!
- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_nonlineareq.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

obj = calculateStandardError(obj,method,draws)

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):
 - stdBeta : Standard deviation of coefficient of the main equation.
 - tStatBeta : T-statistic for the coefficient of the main equation.
 - pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
 - stdLambda : Standard deviation of coefficient of the observation equation.
 - tStatLambda : T-statistic for the coefficient of the observation equation.
 - pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosteriors** ↑

```
[DB,plotter,pAutocorr] = checkPosteriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
 - plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
 - pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments** ↑

c = conditionalTheoreticalMoments(obj,varargin)

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic
- Optional inputs;
- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.
 - 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'draws' must be set to a number > 1.
 - 'method' : The selected method to make the parameter draws. Default is ''. See below for what that means.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mBlockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" paramters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rBlockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" paramters based on estimation on these data.

```

Only an option for models estimated
with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
overlapping random block length
bootstrap., and then "draw" parameters
based on estimation on these data.
Only an option for models estimated
with classical methods.

> 'posterior'          : Posterior draws. Only for models
estimated with bayesian methods.
Default for models estimated with
bayesian methods.

Caution : Posterior draws is
already made at the
estimation stage.

- 'nSteps'      : Number of steps ahead. As an integer.

```

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constraints ↑**

```
[C,Ceq] = nb_model_parse.constraints(pars,constrFuncEq,constrFuncIneq,
varargin)
```

Description:

Function that will be converted into a function handle during estimation when parameter constraints are added.

Input:

- pars : The current values of the parameters.
- constrFuncEq : A function handle, which is constructed in nb_model_parse.constraints2func, that evaluate the equality constraints on the parameters.
- constrFuncIneq : A function handle, which is constructed in nb_model_parse.constraints2func, that evaluate the inequality constraints on the parameters.

Output:

- C : The value of the evaluated inequality constraints. Parameter values are feasible as long as all(C <= 0).
- Ceq : The value of the evaluated equality constraints. Parameter values are feasible as long as all(abs(Ceq) < eps), for some small eps.

See also:

[nb_nonlineareq.constraints2func](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **constructCondDB** ↑

```
condDB           = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s)

to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...  
    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.

- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **convert ↑**

obj = convert(obj,freq,method,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the

input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name, input, shift, description
  'VAR1_G', 'pcn(VAR1)', 'avg', 'VAR1 growth'
  'VAR2_G', 'pcn(VAR2)', 'avg', 'VAR2 growth'
  'VAR3_G', 'pcn(VAR3)', 'avg', 'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Åtterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest ↑**

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_nonLinearEq.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj,draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_nonlineareq.forecast](#), [nb_nonLinearEq.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.

- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!

- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_nonlineareq.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity ↑**

obj = doSimulateFromDensity(obj, draws)

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_nonLinearEq.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments ↑**

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)),

where `x` is the variable along the first dimension. In the lower triangular part the you can find `cov(x(-i),y)`, where `x` is the variable along the first dimension.

E.g: You have two variables `x` and `y`, then to find `cov(x,y(-i))` you can use `getVariable(varargin{i},'y','x')`, while to get `cov(x(-i),y) (== cov(x,y(+i)))` you can use `getVariable(varargin{i},'x','y')`. (This example only works if the output is of class `nb_cs`)

See also:

[nb_nonLinearEq](#)[graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **eq ↑**

`ret = eq(obj,other)`

Description:

Test if `nb_model_generic` objects are equal. In the sens they have the same properties.

Input:

- `obj` : A `nb_model_generic` object either with same size as others, or equal to 1.
- `others` : A `nb_model_generic` model with `dim == 3` or less

Output:

- `ret` : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind      = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **estimate** ↑

```
obj          = estimate(obj,varargin)
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► evalFcstAtDates ↑

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)  
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

obj = evaluateForecast(obj,varargin)

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.

- 'fcstEval' : See the documentation of the same input in the forecast method.

- 'compareToRev' : See the documentation of the same input in the forecast method.

- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsg.e.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► forecast ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' : - '' : No forecast evaluation. Default
- 'se' : Square error forecast evaluations.
- 'abs' : Absolute forecast error
- 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
- 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states

input is providing the full path of regimes. Default is 1.

- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See nb_getDensityPoints (default is that 1000 observations of the density is stored).
 - > integer : The min and max is found but the length of the bins is given by the provided integer.
 - > cell : Must be on the format:
 - {'Var1',lowerLimit1,upperLimit1,binsL1; 'Var2',lowerLimit2,upperLimit2,binsL2; ...}
 - lowerLimit1 : An integer, can be nan, i.e. will be found.
 - upperLimit1 : An integer, can be nan, i.e. will be found.
 - binsL1 : An integer with the length of the bins. Can be nan. I.e. bins length will be adjusted to create a domain of 1000 elements.

Caution : The variables not included will be given the default domain. No warning will be

given if a variable is provided in the cell but not forecast by the model. (This because different models can forecast different variables)

Caution : The combineForecast method of the nb_model_group class is much faster if the lower limit, upper limit! Or else it must simulate new draws and do kernel density estimation for each model again with the shared domain of all models densities.

- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is not empty this will set the start date of the recursive forecast. Default is to start from the first possible date.

- 'endDate' : The end date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used.

Caution: If 'fcstEval' is empty this input will have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).

- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.

- 'condDB' : One of the following:

> A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.

> A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.

> A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also

possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)

> A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the

conditional information. If an shock/residual is not included here, it means that it is not activated.

> StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

> Horizon : Anticipated horizon of the shocks/residual. 1 is default.

> Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.

- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.

- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.

> 'endo' : All the endogenous variables are returned.

> 'fullendo' : All the endogenous variables are returned included the lag variables.

> 'full' : All the variables are returned included the lag variables. Also

including exogenous and shocks.

> 'all' : All variables are returned (but not the lags). Including exogenous and shocks.

- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.

> double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.

> 'mean' : Start from the mean of the data observations. Default.

> 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

> 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)

- 'startingProb' : Either a double or a char:

> [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.

> double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.

> 'ergodic' : Start from the ergodic transition probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'states' : Either an integer with the regime to forecast/simulate in, or an object of class nb_ts with the regimes to condition on. The name of the

variable must be 'states'.

Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!

- 'compareToRev' : Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
 - 'compareTo' : Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size $n \times 2$, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.
 - 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.
 - 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
- Options are:
- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
 - 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.
- Caution :** If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.
- Caution :** If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.
- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1,'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matix. For the order of the variables see; obj.solution.endo. The

value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
- 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_nonLinearEq.parameterDraws](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist ↑**

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_nonlineareq.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► forecastPerc2ParamDist ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_nonlineareq.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst, ...
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditinal information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_nonlineareq.forecast](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

dependentNames = getDependentNames(obj)

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)

Written by Kenneth SÃ¶terhagen Paulsen
```

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- start : An object of class nb_date.
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

```
- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.
```

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsgf.filter](#), [nb_nonlineareq.estimate](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.
- outputType :
 - > 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

```

> 'graph'      : This option can be used to make it easier to plot the
                  recursive densities.

The fcstData output is a struct where the fields
are the recursive forecast periods. Each field has size
nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields
are the recursive forecast periods. Each field has size
nHor x nVars x nPerc. Storing the
percentiles/simulations of the forecast if they is
calculated, or else it is a empty struct.

To graph the recursive forecast use this example:

fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date'       : A string or a nb_date object with the date of the
                  wanted forecast. E.g. '2012Q1'. The fcstData will
                  be a nb_ts object with size nHor x nVar x nPerc + 1,
                  while fcstPercData will be empty. nPerc will then be
                  the number of percentiles/simualtions. Mean will be at
                  last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'    : This option will return the forecast as a
                  nPeriods + nHor x nVar x nHor nb_ts object. This
                  option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
                  options for the 'date' and 'horizon' outputType.
                  Default is false (0).

```

Optional inputs:

```

> 'horizon'    : The fcstData output will be a nb_ts object with
                  size nRec x nVars. While the fcstPercData output
                  will be a nb_ts object with size nRec x nVars x nSim.

```

You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_nonLinearEq.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables** ↑

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars ↑**

```
vars = getLHSVars(obj)
vars = getLHSVars(obj,varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.

- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit           = getPIT(obj)
[pit,plotter] = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.

- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the

start date of the data stored in the model object.

- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods ↑**

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

: Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions** ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.

- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_subplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getRecursiveScore ↑](#)

```
score          = getRecursiveScore(obj,type)  
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores

```

        - 'MLS'      : Mean log score

- dim          : Either 'variables' to get the recursive scores of all
                  the variables forecast by a model (the obj input must
                  be a scalar object), or else the name of the variable
                  to get the score of (must be part of all models!).

- startDate    : The date to begin constructing the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- endDate      : The date to end constructiong the score. Can be
                  empty. Either as a string or an object of class
                  nb_date.

- invert        : true | false. Default true. If true it will invert the
                  score for 'RMSE', 'MSE', 'MAE' and 'STD'.

```

Output:

```

- score       : A nb_ts object with size nPeriods x nVars x nHorizon if dim
                  equal 'variables', else a nb_ts object with size nPeriods x
                  nObj x nHorizon.

- plotter     : A nb_graph_ts object to plot the recursive scores. Use the
                  graph method. One figure per horizon!

```

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual ↑**

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

```
- obj : An object of class nb_model_generic
```

Output:

```
- residual : Either a nb_ts or a nb_cs object with residual(s) stored.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

roots = getRoots(obj)

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if

allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast. These are the variables that may be given to the 'varOfInterest' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'.
- observables : A cellstr with the observables the model may forecast. These are the observables that may be given to the 'observables' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation ↑**

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

`nb_nonLinearEq.simulatedMoments`, `nb_nonLinearEq.theoreticalMoments`
`nb_nonLinearEq.empiricalMoments`, `nb_graphPagesGUI`, `nb_graph_cs`

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **handleCondInfo** ↑

`ret = handleCondInfo(obj)`

Description:

Check if a `nb_model_estimate` object handle conditional info.

Input:

- `obj` : A scalar `nb_model_estimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass `NB_MODEL_ESTIMATE`

► **handleMissing** ↑

`ret = handleMissing(obj)`

Description:

Check if a `nb_model_estimate` object handle missing observations.

Input:

- `obj` : A scalar `nb_model_estimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass `NB_MODEL_ESTIMATE`

► **help** ↑

```
helpText = nb_nonLinearEq.help  
helpText = nb_nonLinearEq.help(option)  
helpText = nb_nonLinearEq.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods' : The number of periods that you let innovations from

the model represented by obj be active.

- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_nonlineareq.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [irf](#) ↑

[irfs,irfsBands,plotter] = irf(obj,varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsg objects.
- 'compare' : Give true if you have a scalar nb_dsg model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.
- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.
- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.

I.e. {'var1',100,...} or
{{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}
- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

```

- 'foundReplic'      : A struct with size 1 x replic. Each element
                      must be on the same format as obj.solution. I.e.
                      each element must be the solution of the model
                      given a set of drawn parameters. See the
                      parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare'       : A struct on the same format as the irfs output from
                      this function with the IRFs to compare to.

- 'levelMethod'       : One of the following:

  > 'cumulative product'           : cumprod(1 + X,1)
  > 'cumulative sum'              : cumsum(X,1)
  > 'cumulative product (log)'     : log(cumprod(1 + X,1))
  > 'cumulative sum (exponential)' : exp(cumsum(X,1))
  > 'cumulative product (%)'       : cumprod(1 + X/100,1)
  > 'cumulative sum (%)'          : cumsum(X/100,1)
  > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
  > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
  > '4 period growth (log approx)' : nb_msum(X,3)

- 'method'             : The selected method to create error bands.
                      Default is ''. See help on the method input of the
                      nb_model_generic.parameterDraws method for more
                      this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent
                  variables. See nb_var.set_identification. Of
                  course this option only makes sense for VARs
                  identified with sign restrictions.

- 'normalize'          : A 1 x 3 cell. First element must be the variable
                      name as a string. The second element must be the
                      value to normalize to, as an integer. While the
                      third element is the period to be normalized, if
                      set to inf, it will normalize the max impact
                      period.

E.g. {'Var',1,2}, {'Var',1,inf}

Caution: 2 means observation 2 of the IRF, and as
          the IRF start at period 0, this means that
          observation 2 is period 1.

- 'normalizeTo'        : 'draws' or 'mean'. 'draws' normalize each draw of
                      irfs, while 'mean' normalize the mean and scale
                      percentiles/other draws accordingly. Default is

```

```

'draws'.

Caution: Setting this to 'mean' will not work for
reported variables that is not measured
as % deviation from steady-state/mean.

- 'newReplic'
  : When out of parameter draws, this factor decides how
many new draws are going to be made. Default is 0.1.
I.e. 1/10 of the replic input.

- 'parallel'
  : Give true if you want to do the irfs in parallel.
This option will parallelize over models. Default
is false.

- 'parallelL'
  : Give true if you want to do the irfs in parallel.
This option will parallelize over parameter
simulations. Only an option if numel(obj) == 1.
Default is false.

- 'pause'
  : true or false, Enable or disable pause option
when calculating the irfs. Default is true.

- 'perc'
  : Error band percentiles. As a 1 x numErrorBand
double. The input must be given as the coverage, and
not the percentiles itself. E.g. [0.3,0.5,0.7,0.9].
0.3 means to return the two percentiles 35 and 65,
which will cover 30% of the distribution around the
median. So [0.3,0.5,0.7,0.9] will get the
percentiles [5,15,25,35,65,75,85,95], i.e. this
will restrict the percentiles to be symmetric. If
empty (default) all the simulation will be returned.
(The graph will give percentiles specified by the
'fanPerc' input (0.68).)

- 'periods'
  : Number of periods of the impulse responses.

- 'plotSS'
  : Set to true to plot the steady-state in the IRF
graphs. Default is false. Only for nb_dsg objects.

- 'plotDevInitSS'
  : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic'
  : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings'
  : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks'
  : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may

```

be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char (Only for Markov-switching models):
 - > scalar : Select the the regime to take the initial transition probabilities from.
 - > double : A draws x nRegime or a 1 x nRegime double. draws refer to the number set by the 'draws' input.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default for 'irf'.
 - > 'random' : Randomize the starting values using the simulated starting points. Default for 'girf'.
- 'startingValues' : Either a double or a char:
 - > double : A nVar x 1 double.
 - > 'steadystate' : Start from the steady state. If you are dealing with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.
 - > 'zero' : Start from zero on all variables.
 - > 'random' : Randomize the starting values using the simulated starting points. Default when 'type' set to 'girf'.
- 'states' : Either an integer with the states to produce irf in, or a double with the states to condition on. Must have size periods x 1 in the last case. Applies to both MS - models and break-point models.
- 'type' : 'girf' or 'irf'. (If empty 'irf' will be used)

```

> 'girf' : Generalized impulse response function
           calculated as the mean difference between
           shocking the model with a one period shock
           (1 std) and no shock at all. Use the 'draws'
           input to set the number of simulations to use
           to base the calculation on. This type of
           irfs must be used for non-linear models.

> 'irf'   : Standard irf for linear models. Calculated by
           shocking the model with a one period shock
           (1 std).

- 'variables'      : The variables to create impulse responses of.
                      Default is the dependent variables defined by the
                      first model.

Caution: The variables reported by the reporting
          property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                      using the method specified by 'levelMethod'.

```

Output:

```

- irfs      : The (central) impulse response function stored in a
               structure of nb_ts object. Each field stores the the
               impulse responses of each shock. Where the variables
               of the nb_ts object is the impulse responses of the
               wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is
      stored in irfs.'(E_X_NW')'. Which will then be a nb_ts
      object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
          responses from each model are saved as
          different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different
          variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store
               a nb_ts object with the error bands of all variables. The
               pages of the nb_ts object is the percentiles (from lower
               to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter   : An object of class nb_graph_ts.

> 'compareShocks' set to false (default)
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

> 'compareShocks' set to true
    Use the graphSubPlots method or the
    nb_graphSubPlotGUI class to produce the graphs.

```

- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_nonLinearEq.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

ret = isDensityForecast(obj)

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

ret = isFiltered(obj)

Description:

Test if a nb_dsgc object is filtered or not.

Input:

- obj : An object of class nb_dsgc.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

ret = isMS(obj)

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsg objects!

For nb_dsg objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

ret = isestimated(obj)

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.
- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_nonLinearEq.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.

- type : The type of restriction. Either 'irf', 'corr' or 'cov'.

- restriction : Depends on the type input:

```

> 'irf'      : A N x 4 cell, where the elements of each
               row are:
               1. Name of the shock. 'E_X'
               2. Name of the variable. 'X'
               3. Horizon. E.g. 1 or 1:3.
               4. Restriction. E.g. @(x)gt(x,0),
                  @(x)gt(x,0)||lt(x,1), i.e. a
                  function_handle that takes a scalar
                  double as input and a scalar logical as
                  output.

> 'rirf'     : A N x 7 cell, where the elements of each
               row are:
               1. Name of the shock 1. 'E_X'
               2. Name of the variable 1. 'X'
               3. Horizon of variable 1. E.g. 1.
               4. Name of the shock 2. 'E_Y'
               5. Name of the variable 2. 'Y'
               6. Horizon of variable 2. E.g. 2.
               7. Restriction. E.g. @(x,y)gt(x,y),
                  @(x,y)gt(x-y,0)||lt(x-y,1), i.e. a
                  function_handle that takes a two scalar
                  double as inputs and a scalar logical
                  as output.

> 'corr'     : A N x 4 cell, where the elements of each
               row are:
               1. Name of the first variable.
               2. Name of the second variable.
               3. Number of periods to lag the 2. variable.
               4. Same as 4 for 'irf'.

               E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to
               test a restriction on the autocorrelation
               at lag for variable 'Var1'.

               E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to
               test a restriction on the contemporaneous
               correlation between 'Var1' and 'Var2'.

> 'cov'       : Same as for 'corr'.

               E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to
               test a restriction on the contemporaneous
               variance.

> 'ss'        : A N x 2 cell, where the elements of each
               row are:
               1. The expression using any of the
                  endogenous variables of the model.
               2. Same as 4 for 'irf'.

```

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

`nb_nonLinearEq.monteCarloFiltering`, `nb_nonlineareq.irf`
`nb_nonLinearEq.theoreticalMoments`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **mincerZarnowitzTest ↑**

`[test,pval,res] = mincerZarnowitzTest(obj,precision)`

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- `obj` : An object of class `nb_model_generic`. You need to call one of the forecasting functions first!
- `precision` : The precision of the printed result. As a string. Default is '`%8.6f`'.

Output:

- `test` : A `nb_ts` object with the test statistic. As a `nHor x nModel x nVar nb_ts` object.
- `pval` : A `nb_ts` object with the p-values of the test. As a `nHor x nModel x nVar nb_ts` object.
- `res` : A char with the printout of the test.

See also:

`nb_nonLinearEq.uncondForecast`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **monteCarloFiltering ↑**

```
paramD           = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if

the model where solved and the test function returned a value.

- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_nonLinearEq.mcfRestriction](#), [nb_nonlineareq.solve](#)
[nb_nonLinearEq.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.

```

> 'blockBootstrap'      : Create artificial data by
                           non-overlapping block bootstrap,
                           and then "draw" parameters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'mblockBootstrap'    : Create artificial data by
                           overlapping block bootstrap,
                           and then "draw" parameters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'rblockBootstrap'    : Create artificial data by
                           overlapping random block length
                           bootstrap, and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
                           overlapping random block length
                           bootstrap., and then "draw" parameters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'copulaBootstrap'    : Uses a copula approach to draw residual
                           that are autocorrelated, Does not handle
                           heteroscedasticity. Only an option for
                           models estimated with classical methods.

> 'posterior'          : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores
 available.
- 'newDraws' : When out of parameter draws, this factor decides how
 many new draws are going to be made. Default is 0.1.
 I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given

as a number less than 1, it will by default be set to 100.

- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:

> 'param' : Default. A struct with the following fields:

 beta : A nPar x nEq x draws double with the estimated parameters.

 sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:

 lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.

 R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.

 factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterIntervals** ↑

ci = parameterIntervals(obj,alpha)

Description:

```
Get confidence intervals (classic) or probability intervals (bayesian).
```

```
Confidence intervals are constructed as:
```

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

```
Probability intervals are constructed as the per
```

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parse** ↑

```
obj = nb_nonLinearEq.parse(filename,varargin)
```

Description:

Parse model file on the .nb format.

Input:

- filename : A string with the name of the model file with extension .nb.

Optional inputs:

- 'macroProcessor' : See nb_nonLinearEq.help('macroProcessor').
- 'macroVars' : See nb_nonLinearEq.help('macroVars').
- 'silent' : See nb_nonLinearEq.help('silent').

Output:

- obj : An object of class nb_nonLinearEq.

See also:

[nb_nonLinearEq.template](#), [nb_nonLinearEq.help](#)

Written by Kenneth Sæterhagen Paulsen

► **parseTempParameters** ↑

```
out = nb_model_parse.parseTempParameters(eqs,verbose)
```

Description:

Parse terms starting with #, and substitute out for where these terms are used.

Input:

- eqs : A N x 1 cellstr with the equations of the model.
- verbose : Set to true to print resulting equations.

Output:

- out : A N x 1 cellstr with the parsed equations of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_PARSE

► **plotForecast** ↑

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.

- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

```
plotter = A nb_graph_data object. Use the graph method to produce the
figure, or the nb_graphPagesGUI class.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► plotMCF ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound, ...
                                     upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_nonLinearEq.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_nonLinearEq.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_nonLinearEq.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosterioriors ↑**

```
plotter = plotPosterioriors(obj)
plotter = plotPosterioriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_nonLinearEq.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPriors](#) ↑

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results** ↑

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_nonlineareq.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

```
- obj      : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA
```

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

```
- obj : A vector of nb_model_estimate objects.
```

Optional input:

If number of inputs equals 1:

```
- varargin{1} : A structure of fields to be set. See the
               template method of each model class for more.
```

Else:

```
- varargin     : ...,'inputName', inputValue,... arguments.
```

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

```
- obj : A vector of nb_model_estimate objects.
```

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition** ↑

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';  
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the

nb_model_generic.parameterDraws method for more this input. An extra option is:

```
> 'identDraws' : Uses the draws of the matrix with  
      the map from structural shocks to dependent variables.  
      See nb_var.set_identification. Of course this option  
      only makes sense for VARs identified with sign  
      restrictions.  
  
- 'replic'          : The number of simulation for posterior, bootstrap and  
      MC methods. Default is 1. Only used when 'perc' is  
      provided.  
  
      Caution: Will not have anything to say for the method  
      'identDraws'. See the option 'draws' of the  
      method nb_var.set_identification for more.  
  
- 'stabilityTest' : Give true if you want to discard the draws  
      that give rise to an unstable model. Default  
      is false.  
  
- 'parallel'        : Give true if you want to do the shock decomp in  
      parallel. Default is false.  
  
- 'fcstDB'          : An object of class nb_ts with the forecast to  
      decompose. must contain all model variables and  
      shocks/residuals, and must start the period after  
      the estimation/filtering end date or at the  
      estimation/filtering start date. See  
      the nb_model_generic.getForecast method.  
  
- 'type'            : Choose 'updated' or 'smoothed'. 'smoothed' is  
      default.  
  
- 'anticipationStartDate' : Start date of anticipating shocks. As a  
      string. If different models has different  
      frequency this date will be converted.  
  
- 'model2'          : A struct with the solution of the second  
      model to use for decomposition.  
  
- 'secondModelStartDate' : Start date of second model. As a string.  
      If different models has different frequency  
      this date will be converted.
```

Output:

- decomp : A structure of nb_ts objects with the shock
 decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty
 bounds of the shock decomposition for each model at each
 percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the
 graph method to produce the graphs or use the
 nb_graphPagesGUI on each element of the graph objects.

See also:

`nb_nonLinearEq.parameterDraws, nb_shockDecomp`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **simulate** ↑

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- `obj` : A vector of `nb_model_generic` objects.
- `nSteps` : Number of simulation steps. As a `1x1 double`. Default is 100.

Optional inputs:

- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- `'bounds'` : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - `'shock'` : Name of the shock to match the restriction on the bounds of the given variable.
 - `'lower'` : The lower bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.
 - `'upper'` : The upper bound of the selected variable. Either a `1x1 double` value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps double` matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a `1x1 double` or a `nSteps x 1 double`.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
- 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty

the simulation will switch between regimes. Only an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsg models.
 - > 'steadyState' : Start from the steady state. For now only an option for nb_dsg models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsg models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consists of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the

'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c]           = simulatedMoments(obj,varargin)
[m,c,ac1]       = simulatedMoments(obj,varargin)
[m,c,ac1,ac2]   = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i},'y','x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i},'x','y'). (This example only works if the output is of class nb_cs)

See also:

[nb_nonLinearEq.graphCorrelation](#), [nb_nonLinearEq.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_nonlineareq.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_nonLinearEq.template()
options = nb_nonLinearEq.template(num)
```

Description:

Construct a struct which can be provided to the nb_nonLinearEq class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```
- obj : A scalar solved nb_model_generic object.  
  
- expression : A MATLAB expression as a string. Use parameter names as  
variables. See model.parameters.name.  
  
- method : A string with the method to use. For bootstrap method see  
nb_bootstrap. For bayesian models 'posterior' is the only  
option. Default is 'bootstrap' for classical models, while  
'posterior' is the default for bayesian models.  
  
- draws : Number of draws from the parameter distribution. Default  
is 1000.  
  
- alpha : Confidence level. Default is 0.05.  
  
- type : Type of test:  
    > '=' : Two sided test. expression == 0 (default)  
            For two-sided tests it is assumed that the  
            distribution is symmetric! Use  
            confidence/probability intervals instead.  
    > '>' : One sided test. expression > 0  
    > '<' : One sided test. expression < 0
```

Output:

```
- pval : > 'classical' : P-value of test.  
        > 'bayesian' : Probability of test.  
  
        A 1x1 double.  
  
- ci : A 1 x 2 double with the lower and upper bound of the  
confidence/probability interval of the tested expression.  
  
- dist : A nb_distribution object storing the distribution of the  
tested expression.
```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c] = theoreticalMoments(obj,varargin)
[m,c,ac1] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2] = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the

parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_nonLinearEq.graphCorrelation](#), [nb_nonLinearEq.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.

- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.

```
> {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object  
    storing the numerically calculated  
    forecast error variances/skewnesses.  
  
- plotter : A nb_graph_ts object which the method graph can be used to  
    produce graphs.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_nonlineareq.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_nonlineareq.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update ↑**

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.

E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_nonLinearEq.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_rw

Go to: [Properties](#) | [Methods](#)

A class for estimation of random walk models.

Superclasses:

nb_model_generic

Constructor:

```
obj = nb_rw(varargin)
```

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_rw object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|-------------------------------|--------------------------------------|-----------------------------|----------------------------------|-----------------------------|
| • addAutoName | • addAutoNameIfLocal | • addID | • addIDIfLocal | • dependent |
| • endogenous | • estOptions | • exogenous | • forecastOutput | • name |
| • options | • parameters | • reporting | • residuals | • results |
| • solution | • transformations | • userData | | |

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsge models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **[endogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that persevere the time span

are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.

- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double. (Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is

for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- `getScore`
- `getVariablesList`
- `handleCondInfo`
- `help`
- `interpretForecast`
- `isDensityForecast`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `printCov`
- `removeObservations`
- `shock_decomposition`
- `simulateFromDensity`
- `solve`
- `solveRW`
- `solveVector`
- `template`
- `testParameters`
- `uncertaintyDecomposition`
- `unstruct`
- `variance_decomposition`
- `getSolution`
- `graphCorrelation`
- `handleMissing`
- `initialize`
- `irf`
- `isFiltered`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `set`
- `simulate`
- `simulatedMoments`
- `solveNormal`
- `solveRecursive`
- `struct`
- `testForecastRestrictions`
- `theoreticalMoments`
- `uncondForecast`
- `update`

► **`appendData`** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters ↑**

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will
be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the
double method on this output to convert it to a double
matrix.

See also:

[nb_rw.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws
(bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see
[nb_bootstrap](#). For bayesian models 'posterior' is the only
option. Default is 'bootstrap' for classical models, while
'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is
1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property
is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convert`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convert`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

`obj = convert(obj,freq,methods,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_rw.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_rw.forecast](#), [nb_rw.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_rw.forecast, nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast, nb_rw.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- `obj` : An object of class `nb_model_generic`

Optional inputs;

- `'vars'` : Either `'dependent'` or a cellstr with the wanted variables. `'dependent'` will return the moment of the dependent variables. `'dependent'` is default.
- `'output'` : Either `'nb_cs'` or `'double'`. Default is `'nb_cs'`.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when `'stacked'` is set to true.
- `'type'` : Either `'covariance'` or `'correlation'`. Default is `'correlation'`.
- `'startDate'` : The start date of the calculations. Default is the start date of the `options.data` property. Only for time-series! Must be a string or a `nb_date` object.
- `'endDate'` : The end date of the calculations. Default is the end date of the `options.data` property. Only for time-series! Must be a string or a `nb_date` object.
- `'demean'` : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- `varargout{1}` : The mean, as a $1 \times nVar$ `nb_cs` object or double.
- `varargout{2}` : The contemporaneous covariances/correlations, as a $nVar \times nVar$ `nb_cs` object or double. The variance is along the diagonal. (Will be symmetric)
- `varargout{i}` : $X > 2$. The auto-covariances/correlations, as a $nVar \times nVar$ `nb_cs` object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find `cov(x,y(-i))`, where `x` is the variable along the first dimension. In the lower triangular part you can find `cov(x(-i),y)`, where `x` is the variable along the first dimension.

E.g: You have two variables `x` and `y`, then to find `cov(x,y(-i))` you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get `cov(x(-i),y)` ($\approx \text{cov}(x,y(+i))$) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class `nb_cs`)

See also:

[nb_rw.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast ↑**

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'          : The length of the bins og the domain of the density.
                    Either;
                      > []      : The domain will be found. See
                        nb_getDensityPoints (default is that 1000
                        observations of the density is stored).
                      > integer : The min and max is found but the length
                        of the bins is given by the provided
                        integer.
                      > cell     : Must be on the format:
                        {'Var1',lowerLimit1,upperLimit1,binsL1;
                         'Var2',lowerLimit2,upperLimit2,binsL2;
                         ...}
                      - lowerLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - upperLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - binsL1      : An integer with the
                        length of the bins. Can
                        be nan. I.e. bins length
                        will be adjusted to
                        create a domain of 1000
                        elements.

Caution : The variables not included will be given
           the default domain. No warning will be
           given if a variable is provided in the
           cell but not forecast by the model. (This
           because different models can forecast
           different variables)

Caution : The combineForecast method of the
           nb_model_group class is much faster if the
           lower limit, upper limit! Or else
           it must simulate new draws and do kernel
           density estimation for each model again
           with the shared domain of all models
           densities.

- 'startDate'    : The start date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
         the start date of the recursive forecast.
         Default is to start from the first possible
         date.

- 'endDate'      : The end date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
- > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
- > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
the ergodic steady-state for
MS-model, and in the last regime
for break-point models.

- 'startingProb' : Either a double or a char:

- > [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the ergodic mean is used.
- > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
- > 'ergodic' : Start from the ergodic transition probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

- 'states' : Either an integer with the regime to forecast/simulate in, or an object of class nb_ts with the regimes to condition on. The name of the variable must be 'states'.

Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!

- 'compareToRev' : Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.

- 'compareTo' : Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a kernel density estimation of the density forecast.

- 'exoProj' : Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_rw.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_rw.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_rw.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_rw.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent ↑**

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a $N \times 3$ cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs \times nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_rw.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData           = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_rw.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getForecastVariables ↑](#)

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as

fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

`nb_model_generic.forecast`, `nb_model_generic.constructScore`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getResidual** ↑

`residual = getResidual(obj)`

Description:

Get residual from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- `residual` : Either a `nb_ts` or a `nb_cs` object with `residual(s)` stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualGraph** ↑

`plotter = getResidualGraph(obj)`

Description:

Get residual graph from the given estimator. The returned will be an object of class `nb_graph`, which you can call the `graphInfoStruct` method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- `obj` : An object of class `nb_model_generic` (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph`. Use the `graphInfoStruct` method or the `nb_graphInfoStructGUI` class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of `nb_model_generic` object

Input:

- `obj` : A vector of `nb_model_generic` objects

Output:

- `residualNames` : A cellstr with the unique residual names of all the models

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method `solve`.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
- 'RMSE' : One over root mean squared error
- 'MSE' : One over mean squared error
- 'MAE' : One over mean absolute error
- 'MEAN' : One over mean error
- 'STD' : One over standard error of the forecast error
- 'ESLS' : Exponential of the sum of the log scores
- 'EELS' : Exponential of the mean of the log scores
- 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.

- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_rw.simulatedMoments](#), [nb_rw.theoreticalMoments](#)
[nb_rw.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_rw.help
helpText = nb_rw.help(option)
helpText = nb_rw.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize ↑**

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given,

otherwise this option does not apply.

- 'optimizer'
 - : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'
 - : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'
 - : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'
 - : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods'
 - : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'
 - : The start date of forecast. Must be a string or an object of class nb_date. If empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'
 - : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'
 - : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'
 - : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights'
 - : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_rw.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgc objects.
- 'compare' : Give true if you have a scalar nb_dsgc model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

```

- 'draws' : Number of draws for calculating girf. Default is
           1000. For Markov switching models this will set the
           number of simulated paths of states. For more see
           the 'type' input.

- 'factor' : A cell array with the factors to multiply the
             irf of the individual model variables.
             I.e. {'var1',100,...} or
                  {'var1','var2'},100,...}

If the string starts with a asterisk you will
multiply all the variables that contain that
string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This option sets the error band percentiles of the
               graph, when the 'perc' input is empty. As a 1 x
               numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
               Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from
                  this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

    > 'cumulative product' : cumprod(1 + X,1)

    > 'cumulative sum' : cumsum(X,1)

    > 'cumulative product (log)' : log(cumprod(1 + X,1))

    > 'cumulative sum (exponential)' : exp(cumsum(X,1))

    > 'cumulative product (%)' : cumprod(1 + X/100,1)

    > 'cumulative sum (%)' : cumsum(X/100,1)

    > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))

    > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))

    > '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
             Default is ''. See help on the method input of the
             nb_model_generic.parameterDraws method for more
             this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
                    the map from structural shocks to dependent

```

variables. See `nb_var.set_identification`. Of course this option only makes sense for VARs identified with sign restrictions.

- `'normalize'` : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to `inf`, it will normalize the max impact period.
E.g. `{'Var',1,2}, {'Var',1,inf}`
Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- `'normalizeTo'` : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- `'newReplic'` : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- `'parallel'` : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- `'parallelL'` : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.
- `'pause'` : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- `'perc'` : Error band percentiles. As a 1 x `numErrorBand` double. The input must be given as the coverage, and not the percentiles itself. E.g. `[0.3,0.5,0.7,0.9]`. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So `[0.3,0.5,0.7,0.9]` will get the percentiles `[5,15,25,35,65,75,85,95]`, i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- `'periods'` : Number of periods of the impulse responses.
- `'plotSS'` : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.

```

- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be
a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws
that give rise to an unstable model. Default
is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):
    > scalar : Select the the regime to take the
initial transition probabilities
from.

    > double : A draws x nRegime or a 1 x
nRegime double. draws refer to
the number set by the 'draws'
input.

    > 'ergodic' : Start from the ergodic transition
probabilities. Default for 'irf'.

    > 'random' : Randomize the starting values
using the simulated starting
points. Default for 'girf'.

- 'startingValues' : Either a double or a char:
    > double : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing

```

with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

```

> 'zero'          : Start from zero on all variables.

> 'random'        : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'         : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'           : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'   : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'       : The variables to create impulse responses of.
                     Default is the dependent variables defined by the
                     first model.

Caution: The variables reported by the reporting
          property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                     using the method specified by 'levelMethod'.

```

Output:

```

- irfs      : The (central) impulse response function stored in a
               structure of nb_ts object. Each field stores the the
               impulse responses of each shock. Where the variables
               of the nb_ts object is the impulse responses of the
               wanted endogenous variables.

               I.e. the impuls responses to the shock 'E_X_NW' is
               stored in irfs.('E_X_NW'). Which will then be a nb_ts
               object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
          responses from each model are saved as
          different datasets (pages) of the nb_ts object.

```

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_rw.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isestimated ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isforecasted ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint

```

prediction bands for
macroeconomic risk management

> 'copula' : Using a copula likelihood approach.

> 'mostlik' : Group the paths based on how
likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the
error bands. If empty (default) all forecasting steps stored
in the object is used.

```

Output:

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

- The data of the nb_ts object has size nSteps x nVars
x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend
on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_rw.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. `@(x)gt(x,0), @(x)gt(x,0)||lt(x,1)`, i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. `@(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. `{'Var1','Var1',1,@(x)gt(x,0.1)}`, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. `{'Var1','Var2',0,@(x)lt(x,0.1)}`, to

test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:

1. The expression using any of the endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_rw.monteCarloFiltering](#), [nb_rw.irf](#)
[nb_rw.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest** ↑

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_rw.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.

- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

function_handle, nb_rw.mcfRestriction, nb_rw.solve
[nb_rw.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'copulaBootstrap' : Uses a copula approach to draw residual that are autocorrelated, Does not handle heteroscedasticity. Only an option for models estimated with classical methods.
 - > 'posterior' : Posterior draws. Only for models estimated with bayesian methods.

Default for models estimated with
bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more on this option.
- stable : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
- lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
- factors : Draws of the factors as a T x nFac x

```

draws double.

> 'solution' : Get the solution of the model for each draw from the
distribution of parameters. The output will be a struct
with size 1 x draws. The struct will have the same
format as the solution property of the underlying
object. I.e. only one output!

> 'object'   : Get the draws as a 1 x draws nb_model_generic object.
Each element will be a model representing a given draw
from the distribution of the parameters. I.e. only one
output!

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + ( +/- nb_distribution.t_icdf(alpha/2) ) * stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```

plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)

```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► plotMCF ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot.

'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- `plotter` : > 'allInOne' : A `nb_graph_cs` object. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
 > 'biplot' : A vector of `nb_graph_data` objects with size equal to the number of pairwise combination of the parameters that can be made. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_rw.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFDistTest ↑**

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `test` : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_rw.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `nameValue` : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_rw.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_rw.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_rw.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties
of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

[decomp, decompBand, plotter] = shock_decomposition(obj, varargin)

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and
the names of the groups. If you have not
listed a shock it will be grouped in a
group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';  
'shock_name_1' , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with
a shock name or a cellstr array with the
shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

Caution: Two special identifiers can be used;
'Initial Conditions' and 'Steady-state'. The
first represent the impact of all shocks that
hit the system before the decomposition/
estimation started. The second summarizes the
impact of steady-state changes on the system,

i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. [0.3, 0.5, 0.7, 0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.

- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_rw.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be

a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the

simulation of.

- > 'endo' : All the endogenous variables are returned.
- > 'fullendo' : All the endogenous variables are returned included the lag variables.
- > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
- > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

- > double : A 1 x nVar double.
- > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
- > 'steadyState' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsge models.

```

> 'zero'           : Start from zero on all variables
                      (Except for the deterministic
                      exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'startingProb'  : Either a double or a char:

    > double        : A nPeriods x nRegime or a 1 x
                      nRegime double. nPeriods refer to
                      the number of recursive periods to
                      forecast.

    > 'ergodic'     : Start from the ergodic transition
                      probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity ↑**

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- `obj` : An object of class nb_model_generic.
- Optional inputs;
- `'output'` : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when 'stacked' is set to true.
- `'vars'` : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- `'type'` : Either 'covariance' or 'correlation'. Default is 'correlation'.
- `'draws'` : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- `'pDraws'` : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- `'perc'` : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- `'nSteps'` : Number of simulation steps. As a 1x1 double. Default is 100.
- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting

values of the simulation. Default is 0.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Default is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_rw.graphCorrelation](#), [nb_rw.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [solve ↑](#)

`obj = solve(obj)`

Description:

Solve estimated model(s) represented by nb_rw object(s).

Input:

- obj : A vector of nb_rw objects.

Output:

- obj : A vector of nb_rw objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_rw.solveNormal(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRW** ↑

```
tempSol = nb_rw.solveRW(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_rw.solveRecursive(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_rw.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

options = nb_rw.template()

Description:

Construct a struct which can be provided to the nb_rw class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► testForecastRestrictions ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.

> variable : The variable(s) to test,
as a string or cell array of strings.

> date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.

> test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillty intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is

```

        'correlation'.

- 'pDraws'      : Number of parameter draws. Default is 1. I.e. to use
                  the estimate parameters.

- 'perc'        : Error band percentiles. As a 1 x numErrorBand double.
                  E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
                  all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method'       : The selected method to create confidenc/probability
                  bands. Default is ''. See help on the method input of
                  the nb_model_generic.parameterDraws method for more
                  this input.

- 'foundReplic'  : A struct with size 1 x pDraws. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

```

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar
 x nVar nb_cs object or double. The variance is along
 the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar
 x nVar nb_cs object or double. Along the diagonal is the
 auto-covariance/correlation with the variable itself. In
 the upper triangular part the you can find cov(x,y(-i)),
 where x is the variable along the first dimension. In
 the lower triangular part the you can find cov(x(-i),y),
 where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find
 cov(x,y(-i)) you can use
 getVariable(varargout{i}, 'y', 'x'),
 while to get cov(x(-i),y) (== cov(x,y(+i))) you
 can use getVariable(varargout{i}, 'x', 'y'). (This
 example only works if the output is of class nb_cs)

See also:

[nb_rw.graphCorrelation](#), [nb_rw.parameterDraws](#)

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

obj = uncondForecast(obj,nSteps,varargin)

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_rw.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [unstruct](#) ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_rw.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update](#) ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type : > '' : Only update data. Default
 > 'estimate' : Update data and estimate.
 > 'solve' : Update data, estimate and solve
 > 'forecast' : Update data, estimate, solve and forecast.

- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp, decompBands, plotter, plotterBands] = ...
    variance_decomposition(obj, varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8, inf].

- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and

the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_rw.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_sa

Go to: [Properties](#) | [Methods](#)

A class for estimation of step ahead (indicator) models. I.e. model on the form

$$y(t+h) = c + b*x(t) + e$$

Where $x(t)$ are exogenous variables of the model, and y is the dependent variable of the model.

Superclasses:

`nb_model_generic`

Constructor:

`obj = nb_sa(varargin)`

Optional input:

- See the `set` method for more on the inputs.
`(nb_model_estimate.set)`

Output:

- `obj` : An `nb_sa` object.

See also:

`nb_model_generic`, `nb_model_estimate.set`, `nb_sa.help`, `nb_sa.template`

Written by Kenneth Sætherhagen Paulsen

Properties:

- | | | | | |
|----------------------------|-----------------------------------|--------------------------|-------------------------------|--------------------------|
| • <code>addAutoName</code> | • <code>addAutoNameIfLocal</code> | • <code>addID</code> | • <code>addIDIfLocal</code> | • <code>dependent</code> |
| • <code>endogenous</code> | • <code>estOptions</code> | • <code>exogenous</code> | • <code>forecastOutput</code> | • <code>name</code> |
| • <code>options</code> | • <code>parameters</code> | • <code>reporting</code> | • <code>residuals</code> | • <code>results</code> |
| • <code>solution</code> | • <code>transformations</code> | • <code>userData</code> | | |

- **`addAutoName`** ↑

Add automatic name (first) to default name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **`addAutoNameIfLocal`** ↑

Add automatic name (first) to local name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **`addID`** ↑

Add identifier to default name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **`addIDIfLocal`** ↑

Add identifier to local name in `getName`. Default is false.

Inherited from superclass `NB_MODEL_NAME`

- **`dependent`** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this property to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t$, $e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t$, $u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.

- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- getScore
- getVariablesList
- handleCondInfo
- help
- interpretForecast
- isDensityForecast
- isMS
- isNB
- isStatic
- isestimated
- issolved
- loadPosterior
- mincerZarnowitzTest
- parameterDraws
- plotForecast
- plotMCF
- plotMCFValues
- plotPriors
- printCov
- removeObservations
- shock_decomposition
- simulateFromDensity
- solve
- solveRecursive
- struct
- testForecastRestrictions
- theoreticalMoments
- uncondForecast
- update
- getSolution
- graphCorrelation
- handleMissing
- initialize
- irf
- isFiltered
- isMixedFrequency
- isStateSpaceModel
- isbayesian
- isforecasted
- jointPredictionBands
- mcfRestriction
- monteCarloFiltering
- parameterIntervals
- plotForecastDensity
- plotMCFDistTest
- plotPosteriors
- print
- print_estimation_results
- set
- simulate
- simulatedMoments
- solveNormal
- solveVector
- template
- testParameters
- uncertaintyDecomposition
- unstruct
- variance_decomposition

► **appendData ↑**

```
obj = appendData(obj, DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► assignParameters ↑

obj = assignParameters(obj,varargin)

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

```

> Two input pairs:
  - 'param' : The names of the assign parameters. As a cellstr vector.
  - 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:
  - 'beta'   : Assign all the estimated parameters of the main equation.
                Must be of same size as obj.results.beta.

  - 'sigma'  : Assign all the estimated parameters of the covariance
                matrix of the main equation. Must be of same size as
                obj.results.sigma.

  - 'lambda' : Assign all the estimated parameters of the observation
                equation. Must be of same size as obj.results.lambda. Only
                for factor models.

  - 'R'      : Assign all the estimated parameters of the covariance
                matrix of the observation equation. Must be of same size
                as obj.results.R. Only for factor models.

```

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

```
obj = assignPosteriorDraws(obj,varargin)
```

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

```

> Two input pairs:
  - 'param' : The names of the assign parameters. As a cellstr vector.
  - 'value' : The values of the assign parameters. As a double nParam x
              nReg x nDraws vector.

> Optional number of input pairs:

```

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!
- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_sa.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

obj = calculateStandardError(obj,method,draws)

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see [nb_bootstrap](#). For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter

distribution. Two-sided test against the null hypothesis of the parameters being 0.

- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► checkModel ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► checkPosteriors ↑

```
[DB,plotter,pAutocorr] = checkPosteriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they were drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
 - plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
 - pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkReporting** ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments** ↑

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.

Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
Default is ''. See below for what that means.

> 'bootstrap' : Create artificial data to bootstrap
the estimated parameters. Default
for models estimated with classical
methods.

> 'wildBootstrap' : Create artificial data by wild
bootstrap, and then "draw" parameters
based on estimation on these data.
Only an option for models estimated

with classical methods.

> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  
            horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type, ...
                                         allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach ↑**

```
obj = convert(obj,freq,methods,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables ↑**

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM nb_modelData object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.
 - > third column : Any expression that can be interpreted by the `nb_ts.createShift` method.
 - > fourth column : Comments
- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- `obj` : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- `plotter` : A NxM nb_graph_ts object with a graph with the de-trending. Use the `graphInfoStruct` method or the `nb_graphInfoStruct` class to produce the graph.

Examples:

```
expressions = {%
  Name,    input,    shift,    description
'VAR1_G',  'pcn(VAR1)', 'avg',   'VAR1 growth'
'VAR2_G',  'pcn(VAR2)', 'avg',   'VAR2 growth'
'VAR3_G',  'pcn(VAR3)', 'avg',   'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest ↑**

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string.

```

Default is '%8.6f'.

- 'bandWidth'      : The selected band width of the frequency zero
                      spectrum estimation. Default is to use a automatic
                      selection criterion. See the 'bandWithCrit' input.
                      Must be set to an integer greater then 0.

- 'bandWithCrit'   : Band width selection criterion. Either:

    > 'nw'    : Newey-West selection method.
                  Default.

    > 'a'     : Andrews selection method. AR(1)
                  specification.

- 'multivariate'  : Set to true to do the multivariate test, i.e. test
                      for equal panel forecast. Default is to test each
                      variable separately.

```

Output:

- test : A nb_ts object with the test statistic. As a
 nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a
 nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_sa.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj, draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_sa.forecast](#), [nb_sa.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist** ↑

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_sa.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **doSimulateFromDensity** ↑

```
obj = doSimulateFromDensity(obj,draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_sa.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments ↑**

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
 E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_sa.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [eq ↑](#)

ret = eq(obj,other)

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj      = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this

option is not supported!

- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also being in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:

```

- 'RMSE'   : Root mean squared error
- 'MSE'    : Mean squared error
- 'MAE'    : Mean absolute error
- 'MEAN'   : Mean error
- 'STD'    : Standard error of the forecast error
- 'ESLS'   : Exponential of the sum of the log scores
- 'EELS'   : Exponential of the mean of the log scores
- 'MLS'    : Mean log score

- density : Set true if you have a density forecast.

- quart   : Set true if you want to compare the quarterly forecasts. This
            option automatically detects and uses the quarterly estimates
            when evaluating.

```

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior** ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
 - 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
 - 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
 - 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
- Only for density forecast:
- 'logScore' : Log score
- Can also be a cellstr with the above listed evaluation types.
- Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.
- 'estDensity' : The method to estimate the density. As a string.
Either:

- 'normal' : Normal density approximation.
- 'kernel' : Kernel density estimation. Default.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
- 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density forecast.

- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'bins' : The length of the bins og the domain of the density. Either;
 - > [] : The domain will be found. See

```

nb_getDensityPoints (default is that 1000
observations of the density is stored).

> integer : The min and max is found but the length
of the bins is given by the provided
integer.

> cell      : Must be on the format:

{'Var1',lowerLimit1,upperLimit1,binsL1;
 'Var2',lowerLimit2,upperLimit2,binsL2;
 ...}

- lowerLimit1 : An integer, can be nan,
i.e. will be found.

- upperLimit1 : An integer, can be nan,
i.e. will be found.

- binsL1      : An integer with the
length of the bins. Can
be nan. I.e. bins length
will be adjusted to
create a domain of 1000
elements.

Caution : The variables not included will be given
the default domain. No warning will be
given if a variable is provided in the
cell but not forecast by the model. (This
because different models can forecast
different variables)

Caution : The combineForecast method of the
nb_model_group class is much faster if the
lower limit, upper limit! Or else
it must simulate new draws and do kernel
density estimation for each model again
with the shared domain of all models
densities.

- 'startDate'   : The start date of forecast. Must be a string or an
object of class nb_date. If empty the estim_end_date
+ 1 will be used.

Caution: If 'fcstEval' is not empty this will set
the start date of the recursive forecast.
Default is to start from the first possible
date.

- 'endDate'     : The end date of forecast. Must be a string or an
object of class nb_date. If empty the estim_end_date
+ 1 will be used.

Caution: If 'fcstEval' is empty this input will
have nothing to say.

- 'saveToFile'   : Logical. Save densities and domains to files. One
file for each model. Default is false (do not).

```

- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be

produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain initial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.

Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.

If not provided. Model stds are used.

 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!
- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only

supported for the nb_var and nb_pitvar classes.

- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.
 This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:
 This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.

```

> 'zero'           : Start from zero on all variables
                     (Except for the deterministic
                     exogenous variables)

- 'startingProb' : Either a double or a char:

    > []            : If the model is filtered the
                     starting value is calculated
                     on filtered transition
                     probabilities. Otherwise the
                     ergodic mean is used.

    > double        : A nPeriods x nRegime or a 1 x
                     nRegime double. nPeriods refer to
                     the number of recursive periods to
                     forecast.

    > 'ergodic'     : Start from the ergodic transition
                     probabilities.

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'states'        : Either an integer with the regime to
                     forecast/simulate in, or an object of class nb_ts
                     with the regimes to condition on. The name of the
                     variable must be 'states'.

Caution: For break-point models this is decided by
          the dates of the breaks, and cannot be set
          by this option!

- 'compareToRev'  : Which revision to compare to. Default is final
                     revision, i.e. []. Give 5 to get fifth revision.
                     must be an integer. Keep in mind that this number
                     should be larger than the nSteps input.

- 'compareTo'     : Sometime you want to evaluate the forecast of one
                     variable against another variable which is not
                     part of the model. E.g. if you use the first release
                     of a variable and want to compare it against the
                     final release. To do this you can give a cellstr
                     with size n x 2, where n is the number of variables
                     to change the the actual observations to test
                     against. {'VAR_1','VAR_FINAL',...}.

- 'estimateDensities' : true or false. true if you want to do a
                      kernel density estimation of the density
                      forecast.

- 'exoProj'        : Tolerate missing exogenous variables by projecting
                     them by a fitted model. Only when the 'condDB'
                     input is empty!

Options are:

- ''   : No projection are done. Error will be

```

given if some exogenous variables are not given any conditional information.
 Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficent 0.8, while 'exoVar2' with a constant term equal to

0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

- 'seed'
: Set simulation seed. Default is `2.0719e+05`.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_sa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_sa.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_sa.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getActual](#) ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.

- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB ↑**

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if

recursive estimation is done. See description for a note on real-time estimation.

- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_sa.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by `nb_var.priorTemplate('dsge')`.

Input:

- obj : An object of class nb_model_generic.
- dsgeVar : A nb_var object set up with options for estimation of a DSGE-VAR.

Optional input:

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

dependent = getDependent(obj)

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getFiltered** ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_sa.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0, -1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is

returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!

Caution : If the horizon input is set you will set nHor to one, and the fcstPercData will be non-empty, if density forecast has been produced.

- includeHist : Give true (1) to include history in the output. Only an options for the 'date' and 'horizon' outputType. Default is false (0).

Optional inputs:

> 'horizon' : The fcstData output will be a nb_ts object with size nRec x nVars. While the fcstPercData output will be a nb_ts object with size nRec x nVars x nSim. You can set the horizon to return by the optional input 'horizon'. See the 'timing' option also.

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_sa.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

vars = getForecastVariables(obj)

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► getHistory ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getLHSVars ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj,varsIn)
```

Description:

Get all left hand side variables of the model.
For factor models the observable (+ observableFast) are added as well.
The list will be sorted.
Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

names = getModelNames(obj)

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgen object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getOriginalVariables ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► getPIT ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.
- Optional input
 - varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).
 - : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
 - : Give 'double' to return the value as a numerical array.
 - : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

parameters

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions** ↑

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph ↑**

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore ↑**

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual ↑**

residual = getResidual(obj)

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph ↑**

plotter = getResidualGraph(obj)

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursivly estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.
- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report

the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!

- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast. These are the variables that may be given to the 'varOfInterest' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'.
- observables : A cellstr with the observables the model may forecast. These are the observables that may be given to the 'observables' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'. Only some specific class of models.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation ↑**

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or

`nb_model_generic.theoreticalMoments` (`THEO`), the second input must be the `c` output from `nb_model_generic.empiricalMoments` (`EMP`), the third input can be the `ac1` output from `SIM` or `THEO`, the fourth input can be the `ac1` output from `EMP`, and so on.

Caution : The inputs must be given as `nb_cs` objects.

Caution : Inputs must come in pairs.

Output:

- `plotter` : An object of class `nb_graph_cs`. Use the `graph` method or the `nb_graphPagesGUI` class.

See also:

`nb_sa.simulatedMoments`, `nb_sa.theoreticalMoments`
`nb_sa.empiricalMoments`, `nb_graphPagesGUI`, `nb_graph_cs`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **handleCondInfo** ↑

`ret = handleCondInfo(obj)`

Description:

Check if a `nb_model_estimate` object handle conditional info.

Input:

- `obj` : A scalar `nb_model_etimate` object.

Output:

- `ret` : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_ESTIMATE`

► **handleMissing** ↑

`ret = handleMissing(obj)`

Description:

```
Check if a nb_model_estimate object handle missing observations.
```

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

► **help** ↑

```
helpText = nb_sa.help
helpText = nb_sa.help(option)
helpText = nb_sa.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.

- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_sa.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [irf ↑](#)

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsg objects.
- 'compare' : Give true if you have a scalar nb_dsg model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.
- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.
- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

```

- 'factor'           : A cell array with the factors to multiply the
                      irf of the individual model variables.
                      I.e. {'var1',100,...} or
                           {'var1','var2'},100,...}

                      If the string starts with a asterisk you will
                      multiply all the variables that contain that
                      string with the given factor.
                      E.g. {'*_GAP',100}

- 'fanPerc'         : This options sets the error band percentiles of the
                      graph, when the 'perc' input is empty. As a 1 x
                      numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
                      Default is 0.68.

- 'foundReplic'     : A struct with size 1 x replic. Each element
                      must be on the same format as obj.solution. I.e.
                      each element must be the solution of the model
                      given a set of drawn parameters. See the
                      parameterDraws method of the nb_model_generic class.

                      Caution: You still need to set 'parameterDraws'
                      to the number of draws you want to do.

- 'irfCompare'      : A struct on the same format as the irfs output from
                      this function with the IRFs to compare to.

- 'levelMethod'      : One of the following:

  > 'cumulative product'          : cumprod(1 + X,1)
  > 'cumulative sum'             : cumsum(X,1)
  > 'cumulative product (log)'    : log(cumprod(1 + X,1))
  > 'cumulative sum (exponential)' : exp(cumsum(X,1))
  > 'cumulative product (%)'       : cumprod(1 + X/100,1)
  > 'cumulative sum (%)'          : cumsum(X/100,1)
  > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
  > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
  > '4 period growth (log approx)' : nb_msum(X,3)

- 'method'            : The selected method to create error bands.
                      Default is ''. See help on the method input of the
                      nb_model_generic.parameterDraws method for more
                      this input. An extra option is:

  > 'identDraws' : Uses the draws of the matrix with
                  the map from structural shocks to dependent
                  variables. See nb_var.set_identification. Of
                  course this option only makes sense for VARs
                  identified with sign restrictions.

```

- 'normalize' : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to inf, it will normalize the max impact period.
 E.g. {'Var',1,2}, {'Var',1,inf}
 Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of irfs, while 'mean' normalize the mean and scale percentiles/other draws accordingly. Default is 'draws'.
 Caution: Setting this to 'mean' will not work for reported variables that is not measured as % deviation from steady-state/mean.
- 'newReplic' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- 'parallel' : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.
 Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.
 For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching models):

- > scalar : Select the the regime to take the initial transition probabilities from.
- > double : A draws x nRegime or a 1 x nRegime double. draws refer to the number set by the 'draws' input.
- > 'ergodic' : Start from the ergodic transition probabilities. Default for 'irf'.
- > 'random' : Randomize the starting values using the simulated starting points. Default for 'girf'.

- 'startingValues' : Either a double or a char:
 > double : A nVar x 1 double.
 > 'steadystate' : Start from the steady state. If you are dealing with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean

(default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

- > 'zero' : Start from zero on all variables.
- > 'random' : Randomize the starting values using the simulated starting points. Default when 'type' set to 'girf'.
- 'states' : Either an integer with the states to produce irf in, or a double with the states to condition on. Must have size periods x 1 in the last case. Applies to both MS - models and break-point models.
- 'type' : 'girf' or 'irf'. (If empty 'irf' will be used)
 - > 'girf' : Generalized impulse response function calculated as the mean difference between shocking the model with a one period shock (1 std) and no shock at all. Use the 'draws' input to set the number of simulations to use to base the calculation on. This type of irfs must be used for non-linear models.
 - > 'irf' : Standard irf for linear models. Calculated by shocking the model with a one period shock (1 std).
- 'variables' : The variables to create impulse responses of. Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.
- I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.
- Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.
- Caution: Each nb_model_generic object can have different variables and shocks.
- irfsBands : A struct with the shocks as fieldnames. Each field store

a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_sa.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

ret = isDensityForecast(obj)

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered ↑**

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted** ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.

- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_sa.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. `@(x)gt(x,0), @(x)gt(x,0)||lt(x,1)`, i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. `@(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. `{'Var1','Var1',1,@(x)gt(x,0.1)}`, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. `{'Var1','Var2',0,@(x)lt(x,0.1)}`, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.
 - > 'cov' : Same as for 'corr'.

E.g. `{'Var1','Var1',0,@(x)lt(x,0.1)}`, to test a restriction on the contemporaneous

variance.

```
> 'ss'      : A N x 2 cell, where the elements of each
               row are:
               1. The expression using any of the
                  endogenous variables of the model.
               2. Same as 4 for 'irf'.
```

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_sa.monteCarloFiltering](#), [nb_sa.irf](#)
[nb_sa.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mincerZarnowitzTest ↑**

```
[test,pval,res] = mincerZarnowitzTest(obj,precision)
```

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_sa.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is specified with some lower and upper bounds. For each draw the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.

```

- 'waitbar'      : true or false. Default is true.

- 'output'       : Either 'double' or 'logical'. Default is 'logical'.

- 'seed'         : Set the seed to use to draw the monte carlo simulated
                    parameter sets. Default is 1. Seed is set back to old
                    state after this method is called!

```

Output:

```

- paramD        : The draws made from the parameter space. As a draws x N
                  double. Use paramD(success,:) to get the accepted
                  draws.

- values         : A N x 1 logical/double. An element is true/not nan if
                  the model where solved and the test function returned
                  a value.

- solvingFailed : A N x 1 logical. An element is true if the model could
                  not be solved.

- objAcc        : A 1 x nAcc vector of nb_model_generic objects
                  representing the accepted models.

```

See also:

[function_handle](#), [nb_sa.mcfRestriction](#), [nb_sa.solve](#)
[nb_sa.assignParameters](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **parameterDraws** ↑

```

out = parameterDraws(obj)
out = parameterDraws(obj, draws, method, output, stable)

```

Description:

Make either posterior draws or bootstrapped draws from the distribution
of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

```

- obj      : An object of a subclass of the nb_model_generic class.

- draws    : Number of draws, as an integer. Default is 1000.

- method   : The selected method to make draws.

> 'asymptotic'          : Draw from the confidence set using

```

the assumption of asymptotic normality.
 See the `nb_mlEstimator.drawParameters`
 for more on this option. Applies to models
 estimated with maximum likelihood.

> 'bootstrap' : Create artificial data to bootstrap
 the estimated parameters. Default
 for models estimated with classical
 methods.

> 'wildBootstrap' : Create artificial data by wild
 bootstrap, and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'blockBootstrap' : Create artificial data by
 non-overlapping block bootstrap,
 and then "draw" paramters based
 on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'mblockBootstrap' : Create artificial data by
 overlapping block bootstrap,
 and then "draw" paramters based
 on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'rblockBootstrap' : Create artificial data by
 overlapping random block length
 bootstrap, and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
 overlapping random block length
 bootstrap., and then "draw" paramters
 based on estimation on these data.
 Only an option for models estimated
 with classical methods.

> 'copulaBootstrap' : Uses a copula approach to draw residual
 that are autocorrelated, Does not handle
 heteroscedasticity. Only an option for
 models estimated with classical methods.

> 'posterior' : Posterior draws. Only for models
 estimated with bayesian methods.
 Default for models estimated with
 bayesian methods.

- `output` : 'param' | 'solution' | 'object'. See output section for more
 on this option.
- `stable` : Give true if you want to discard the parameter draws
 that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
 - lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
 - factors : Draws of the factors as a T x nFac x draws double.
- > 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

```
> 'object'      : Get the draws as a 1 x draws nb_model_generic object.  
                  Each element will be a model representing a given draw  
                  from the distribution of the parameters. I.e. only one  
                  output!
```

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```
plotter          = plotForecast(obj)  
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate,...  
                                         increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model
in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the

provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF** ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

```
- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method  
or the nb_graphPagesGUI class to produce the  
graphs.  
> 'biplot' : A vector of nb_graph_data objects with size  
equal to the number of pairwise combination of  
the parameters that can be made. Use the  
graph method or the nb_graphMultiGUI class to  
produce the graphs.
```

See also:

[nb_sa.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest ↑**

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_sa.monteCarloFiltering](#), [nb_cuconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_sa.monteCarloFiltering](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_sa.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

printed = printCov(obj)

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_sa.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

obj = removeObservations(obj, numPer)

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

obj = set(obj,varargin)

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1','...','shock_group_name_N';  
 'shock_name_1' ,..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If

different models has different frequency this date will be converted.

- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_sa.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
 - 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also

including exogenous and shocks.

- > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.
- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a specific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:
 - > double : A 1 x nVar double.
 - > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadystate(state)'. E.g. to start in state 1 give; 'steadystate(1)'. Default for nb_dsge models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:

```

> double      : A nPeriods x nRegime or a 1 x
    nRegime double. nPeriods refer to
    the number of recursive periods to
    forecast.

> 'ergodic'   : Start from the ergodic transition
    probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity** ↑

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```

[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)

```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.
 - 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
 - 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
 - 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
 - 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.
- Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
 - 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
 - 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Defualt is

true.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_sa.graphCorrelation](#), [nb_sa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_sa object(s).

Input:

- obj : A vector of nb_sa objects.

Output:

- obj : A vector of nb_sa objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_sa.solveNormal(results,opt)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_sa.solveRecursive(results,opt)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_sa.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template ↑**

```
options = nb_sa.template()
options = nb_sa.template(num)
```

Description:

Construct a struct which can be provided to the nb_sa class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions ↑**

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► testParameters ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic
is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillity intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the

parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargout{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_sa.graphCorrelation](#), [nb_sa.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.

- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1' , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.

```
> {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object  
storing the numerically calculated  
forecast error variances/skewnesses.  
  
- plotter : A nb_graph_ts object which the method graph can be used to  
produce graphs.
```

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_sa.forecast](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_sa.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **update ↑**

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double.

E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_sa.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

■ nb_singleEq

Go to: [Properties](#) | [Methods](#)

A class for estimation of single equation models.

Superclasses:

nb_model_generic

Constructor:

obj = nb_singleEq(varargin)

Optional input:

- See the set method for more on the inputs.
(nb_model_estimate.set)

Output:

- obj : An nb_singleEq object.

See also:

[nb_model_generic](#), [nb_model_estimate.set](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | | |
|-------------------------------|--------------------------------------|-----------------------------|----------------------------------|-----------------------------|
| • addAutoName | • addAutoNameIfLocal | • addID | • addIDIfLocal | • dependent |
| • endogenous | • estOptions | • exogenous | • forecastOutput | • name |
| • options | • parameters | • reporting | • residuals | • results |
| • solution | • transformations | • userData | | |

- **[addAutoName](#)** ↑

Add automatic name (first) to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addAutoNameIfLocal](#)** ↑

Add automatic name (first) to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addID](#)** ↑

Add identifier to default name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[addIDIfLocal](#)** ↑

Add identifier to local name in getName. Default is false.

Inherited from superclass NB_MODEL_NAME

- **[dependent](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsge models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **[endogenous](#)** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** ↑

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** ↑

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** ↑

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** ↑

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that persevere the time span

are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsgc models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.

- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double. (Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation. As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is

for comments.

All methods of the nb_math_ts class that preserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- appendData
- assignPosteriorDraws
- calculateMultipliers
- checkModel
- checkReporting
- constructCondDB
- convert
- createVariables
- doForecastPerc2Dist
- doSimulateFromDensity
- eq
- evalFctstAtDates
- evaluatePrior
- forecastPerc2Dist
- getActual
- getDSGEVARPriorMoments
- getDependentNames
- getEstimationStartDate
- getForecast
- getHistory
- getModelNames
- getOriginalVariables
- getParameterDrawsMethods
- getPosteriorDistributions
- getRecursiveEstimationGraph
- getResidual
- getResidualNames
- assignParameters
- assignTexNames
- calculateStandardError
- checkPosteriors
- conditionalTheoreticalMoments
- constructScore
- convertEach
- dieboldMarianoTest
- doForecastPerc2ParamDist
- empiricalMoments
- estimate
- evaluateForecast
- forecast
- forecastPerc2ParamDist
- getCondDB
- getDependent
- getEstimationOptions
- getFiltered
- getForecastVariables
- getLHSVars
- getModelVars
- getPIT
- getParameters
- getPredicted
- getRecursiveScore
- getResidualGraph
- getRoots

- getScore
 - getVariablesList
 - handleCondInfo
 - help
 - interpretForecast
 - isDensityForecast
 - isMS
 - isNB
 - isStatic
 - isestimated
 - issolved
 - loadPosterior
 - mincerZarnowitzTest
 - parameterDraws
 - plotForecast
 - plotMCF
 - plotMCFValues
 - plotPriors
 - print Cov
 - removeObservations
 - shock_decomposition
 - simulateFromDensity
 - solve
 - solveOLSEq
 - solveStepAhead
 - solveTSLSEqRecursiv
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
 - getSolution
 - graphCorrelation
 - handleMissing
 - initialize
 - irf
 - isFiltered
 - isMixedFrequency
 - isStateSpaceModel
 - isbayesian
 - isforecasted
 - jointPredictionBands
 - mcfRestriction
 - monteCarloFiltering
 - parameterIntervals
 - plotForecastDensity
 - plotMCFDistTest
 - plotPosteriors
 - print
 - print_estimation_results
 - set
 - simulate
 - simulatedMoments
 - solveNormal
 - solveRecursive
 - solveTSLSEq
 - solveVector
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
-

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **assignParameters** ↑

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta' : Assign all the estimated parameters of the main equation. Must be of same size as obj.results.beta.
- 'sigma' : Assign all the estimated parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma.
- 'lambda' : Assign all the estimated parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models.
- 'R' : Assign all the estimated parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj,varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.
- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.

- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

mult = calculateMultipliers(obj,varargin)

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!

- 'instrument' : A one line char with the name of the instrument. Must be provided!

- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!

- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!

- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.

- 'perc' : Error band percentiles. As a 1 x nPerc double.
E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_singleeq.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError ↑**

```
obj = calculateStandardError(obj,method,draws)
```

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see [nb_bootstrap](#). For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.

Extra outputs (all based on bootstrapped draws):

- stdBeta : Standard deviation of coefficient of the main equation.
- tStatBeta : T-statistic for the coefficient of the main equation.
- pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- stdLambda : Standard deviation of coefficient of the observation equation.
- tStatLambda : T-statistic for the coefficient of the observation equation.
- pValLambda : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
- ci : Confidence interval/probability interval calculated based on a kernel density estimation instead of assuming asymptotic normal distribution. Only for the parameter in beta! As a (nCoeff * nEq) + 1 x 3 cell array.
- dist : Estimated distribution of the coefficients of the main equation. As a (nCoeff * nEq) x 1 nb_distribution object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

`obj = checkModel(obj)`

Description:

Secure that the object is up to date when it comes to the options, estOptions and results struct properties.

Input:

- obj : An object of class nb_model_generic.

Output:

- obj : An object of class nb_model_generic.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosterioriors** ↑

```
[DB,plotter,pAutocorr] = checkPosterioriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they where drawn. This to check for problems with posterior draws that may be autocorrelated.

Input:

- `obj` : A scalar object of class `nb_model_generic`. It must represent a bayesian model that is estimated.

Optional input:

- `'nLags'` : Number of lags to include in the autocorrelation plot. Default is 10.
- `'iter'` : The recursive iteration date. Either as a string or a `nb_date` object. If recursive estimation is done, this input must be provided.

Output:

- `DB` : A `nb_data` object with size `nDraws x numCoeff`.
- `plotter` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.
- `pAutocorr` : A `nb_graph_data` object. Use the `graphSubPlots` or the `nb_graphSubPlotGUI` class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **checkReporting** ↑

```
obj = checkReporting(obj)
```

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **conditionalTheoreticalMoments ↑**

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.

Default is ''. See below for what that means.

> 'bootstrap'	: Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
> 'wildBootstrap'	: Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'blockBootstrap'	: Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'mBlockBootstrap'	: Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'rBlockBootstrap'	: Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'wildBlockBootstrap'	: Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
> 'posterior'	: Posterior draws. Only for models estimated with bayesian methods. Default for models estimated with bayesian methods.
	Caution : Posterior draws is already made at the estimation stage.
- 'nSteps'	: Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...)
                           horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructScore** ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type,...)
                    allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [convert ↑](#)

`obj = convert(obj,freq,method,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► [convertEach ↑](#)

`obj = convert(obj,freq,methods,varargin)`

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of `nb_ts.convertEach`.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to `options.data` and not the `dataOrig` property.

Output:

- See the documentation of `nb_ts.convertEach`.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass `NB_MODELDATA`

► `createVariables` ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- `obj` : A NxM `nb_modelData` object
- `expressions` : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

> second column : Any expression that can be interpreted by the `nb_ts.createVariable` method.

> third column : Any expression that can be interpreted by the `nb_ts.createShift` method.

> fourth column : Comments

- `fcstHorizon` : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```
expressions = { % Name,    input,    shift,    description
  'VAR1_G',    'pcn(VAR1)', 'avg',    'VAR1 growth'
  'VAR2_G',    'pcn(VAR2)', 'avg',    'VAR2 growth'
  'VAR3_G',    'pcn(VAR3)', 'avg',    'VAR3 growth'};

model = model.createVariables(expressions)
```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► dieboldMarianoTest ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.
- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band with selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_singleEq.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

```
obj = doForecastPerc2Dist(obj,draws)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_singleeq.forecast](#), [nb_singleEq.forecastPerc2Dist](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **doForecastPerc2ParamDist ↑**

[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- obj : A nb_model_forecast object.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_singleeq.forecast`, `nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity** ↑

`obj = doSimulateFromDensity(obj, draws)`

Description:

Simulates from the estimated kernel density.

Input:

- `obj` : A `nb_model_forecast` object.
- `draws` : Number of draws to use for simulation.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_model_generic.forecast`, `nb_singleEq.simulateFromDensity`

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **empiricalMoments** ↑

```
[m,c]           = empiricalMoments(obj,varargin)
[m,c,ac1]       = empiricalMoments(obj,varargin)
[m,c,ac1,ac2]   = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the `nb_model_generic` object. The `options.data` property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_singleEq.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **eq** ↑

```
ret = eq(obj,other)
```

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind    = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **estimate** ↑

```
obj        = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default

is to open max number of cores available. Only an option if 'parallel' is given.

- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- density : Set true if you have a density forecast.
- quart : Set true if you want to compare the quarterly forecasts. This option automatically detects and uses the quarterly estimates when evaluating.

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evalaute forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluatePrior ↑**

logPriorD = nb_model_generic.evaluatePrior(prior,par)

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecast** ↑

```
obj          = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only for forward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
- 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error

Only for density forecast:

- 'logScore' : Log score

Can also be a cellstr with the above listed evaluation types.

Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters

for the whole sample.

- 'estDensity' : The method to estimate the density. As a string.
Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
 - 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false.
 - 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
 - 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1, i.e. only produce point forecast.
 - 'regimeDraws' : Number of drawn regime paths when doing density forecast. This will have no effect if the states input is providing the full path of regimes. Default is 1.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. Default is 0.9.
- If set to empty, all simulations will be returned.
- Caution: 'draws' must be set to produce density forecast.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.

```

- 'bins'          : The length of the bins og the domain of the density.
                    Either;
                      > []      : The domain will be found. See
                        nb_getDensityPoints (default is that 1000
                        observations of the density is stored).
                      > integer : The min and max is found but the length
                        of the bins is given by the provided
                        integer.
                      > cell     : Must be on the format:
                        {'Var1',lowerLimit1,upperLimit1,binsL1;
                         'Var2',lowerLimit2,upperLimit2,binsL2;
                         ...}
                      - lowerLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - upperLimit1 : An integer, can be nan,
                        i.e. will be found.
                      - binsL1      : An integer with the
                        length of the bins. Can
                        be nan. I.e. bins length
                        will be adjusted to
                        create a domain of 1000
                        elements.

Caution : The variables not included will be given
           the default domain. No warning will be
           given if a variable is provided in the
           cell but not forecast by the model. (This
           because different models can forecast
           different variables)

Caution : The combineForecast method of the
           nb_model_group class is much faster if the
           lower limit, upper limit! Or else
           it must simulate new draws and do kernel
           density estimation for each model again
           with the shared domain of all models
           densities.

- 'startDate'    : The start date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
         the start date of the recursive forecast.
         Default is to start from the first possible
         date.

- 'endDate'      : The end date of forecast. Must be a string or an
                    object of class nb_date. If empty the estim_end_date
                    + 1 will be used.

Caution: If 'fcstEval' is empty this input will

```

have nothing to say.

- 'saveToFile' : Logical. Save densities and domains to files. One file for each model. Default is false (do not).
- 'observables' : A cellstr with the observables you want the forecast of. Only for factor models. Will be discarded for all other types of models.
- 'condDB' : One of the following:
 - > A nb_ts object with size nSteps x nCondVar with the information to condition on.

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.
 - > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
 - > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
 - > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous

variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is chosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.
- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
If not provided. Model stds are used.
 - > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
 - > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for

all periods. Must be given!

- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogenous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogenous variables are autocorrelated all variables for all periods must be drawn at the same time, using this autocorrelation matrix.
- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:

This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime

is not given, you will start in
the ergodic steady-state for
MS-model, and in the last regime
for break-point models.

- 'startingProb' : Either a double or a char:
 - > 'zero' : Start from zero on all variables
(Except for the deterministic
exogenous variables)
 - > [] : If the model is filtered the
starting value is calculated
on filtered transition
probabilities. Otherwise the
ergodic mean is used.
 - > double : A nPeriods x nRegime or a 1 x
nRegime double. nPeriods refer to
the number of recursive periods to
forecast.
 - > 'ergodic' : Start from the ergodic transition
probabilities.
- 'stabilityTest' : Give true if you want to discard the parameter draws
that give rise to an unstable model. Default
is false.
- 'states' : Either an integer with the regime to
forecast/simulate in, or an object of class nb_ts
with the regimes to condition on. The name of the
variable must be 'states'.

Caution: For break-point models this is decided by
the dates of the breaks, and cannot be set
by this option!

- 'compareToRev' : Which revision to compare to. Default is final
revision, i.e. []. Give 5 to get fifth revision.
must be an integer. Keep in mind that this number
should be larger than the nSteps input.
- 'compareTo' : Sometime you want to evaluate the forecast of one
variable against another variable which is not
part of the model. E.g. if you use the first release
of a variable and want to compare it against the
final release. To do this you can give a cellstr
with size n x 2, where n is the number of variables
to change the the actual observations to test
against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities' : true or false. true if you want to do a
kernel density estimation of the density
forecast.
- 'exoProj' : Tolerate missing exogenous variables by projecting
them by a fitted model. Only when the 'condDB'
input is empty!

Options are:

- '' : No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.

- 'ar' : The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.

Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are beeing produced recursivly with recursivly estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursivly estimated AR cofficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assign to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummie variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example

{'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when `exoProj` is set to 'AR'.

- 'bounds'
: A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a `nVar x nSteps` double matrix. For the order of the variables see; `obj.solution.endo`. The value return by the function must be either a 1x1 double or a `nSteps x 1` double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs'
: This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.

- 'foundReplic'
: A struct with size `1 x parameterDraws`. Each element must be on the same format as `obj.solution`. I.e. each element must be the solution of the model given a set of drawn parameters. See the `parameterDraws` method of the `nb_model_generic` class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.

```
- 'seed' : Set simulation seed. Default is 2.0719e+05.
```

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_singleEq.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_singleeq.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_singleeq.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
    nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.
- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getCondDB ↑

[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provied recursive conditional information.

Caution : If the dataset is of type real-time, this function interpret the endDate input as the estimation end date for the last recursion. It also assumes that each vintage give rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default is false.
- recStart : An object of class nb_date or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if recursive is set to true.

Output:

- condDB : As a nb_ts or nb_data object. Can be given as input to the condDB option of the nb_model_generic.forecast method.
- obj : An object of class nb_model_generic. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the options.data option may change.

See also:

[nb_modelData.createVariables](#), [nb_singleeq.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDSGEVARPriorMoments ↑**

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by [nb_var.priorTemplate\('dsge'\)](#).

Input:

- `obj` : An object of class `nb_model_generic`.
- `dsgeVar` : A `nb_var` object set up with options for estimation of a DSGE-VAR.

Optional input:

- `'maxIter'` : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- `'tol'` : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- `GammaYY` : A $N \times N$ double with the nonstandardized sample moments of the left-hand variables of the VAR.
- `GammaXX` : A $(C + N*L) \times (C + N*L)$ double with the nonstandardized sample moments of the right-hand variables of the VAR.
- `GammaXY` : A $(C + N*L) \times N$ double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getDependent ↑**

`dependent = getDependent(obj)`

Description:

Get dependent variables from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getEstimationOptions** ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► **getEstimationStartDate** ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be $N(0,1)$. Default is false.
- econometricians : Get econometricians view of the filtered variables.
I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a $N \times 3$ cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs \times nvars included the filtered variables.

See also:

[nb_dsge.filter](#), [nb_singleeq.estimate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast** ↑

```
fcstData          = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();
```

```

    end

> 'date'      : A string or a nb_date object with the date of the
                 wanted forecast. E.g. '2012Q1'. The fcstData will
                 be a nb_ts object with size nHor x nVar x nPerc + 1,
                 while fcstPercData will be empty. nPerc will then be
                 the number of percentiles/simualtions. Mean will be at
                 last page.

For real-time forecast the vintage at the time is
returned as the historical data, when includeHist
is true.

> 'horizon'   : This option will return the forecast as a
                 nPeriods + nHor x nVar x nHor nb_ts object. This
                 option will only return the mean forecast.

If includeHist is true the actual data is added
as an additional page of the nb_ts object.

Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).

```

Optional inputs:

```

> 'horizon'   : The fcstData output will be a nb_ts object with
                 size nRec x nVars. While the fcstPercData output
                 will be a nb_ts object with size nRec x nVars x nSim.
                 You can set the horizon to return by the optional
                 input 'horizon'. See the 'timing' option also.

```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_singleEq.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► [getForecastVariables ↑](#)

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object

- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.

- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.

The same apply for real-time data.

- notSmoothed : Prevent getting history from smoothed estimates.

- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

```
names = getModelNames(obj)
```

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```
pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)
```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produced it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

[nb_calculatePIT](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

methods = parameterDraws(obj)

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

p = getParameters(obj,varargin)

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as

fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the parameter values ('struct' is not supported if numel(obj) > 1).

- : Give 'headers' to add model names as headers for each column of the return cell. Only if numel(obj) > 1.
- : Give 'double' to return the value as a numerical array.
- : Give 'skipBreaks' to not remove any break-point parameters.

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted** ↑

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRecursiveScore** ↑

```
score          = getRecursiveScore(obj,type)
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

`nb_model_generic.forecast`, `nb_model_generic.constructScore`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **getResidual** ↑

`residual = getResidual(obj)`

Description:

Get residual from a estimated `nb_model_generic` object

Input:

- `obj` : An object of class `nb_model_generic`

Output:

- `residual` : Either a `nb_ts` or a `nb_cs` object with `residual(s)` stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualGraph** ↑

`plotter = getResidualGraph(obj)`

Description:

Get residual graph from the given estimator. The returned will be an object of class `nb_graph`, which you can call the `graphInfoStruct` method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- `obj` : An object of class `nb_model_generic` (or a subclass of this class).

Output:

- `plotter` : An object of class `nb_graph`. Use the `graphInfoStruct` method or the `nb_graphInfoStructGUI` class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Åström Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getResidualNames** ↑

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of `nb_model_generic` object

Input:

- `obj` : A vector of `nb_model_generic` objects

Output:

- `residualNames` : A cellstr with the unique residual names of all the models

Written by Kenneth Åström Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method `solve`.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getScore ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,...  
    rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
- 'RMSE' : One over root mean squared error
- 'MSE' : One over mean squared error
- 'MAE' : One over mean absolute error
- 'MEAN' : One over mean error
- 'STD' : One over standard error of the forecast error
- 'ESLS' : Exponential of the sum of the log scores
- 'EELS' : Exponential of the mean of the log scores
- 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.

- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursivly using the the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution ↑**

value = getSolution(obj,type)

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the 'varOfInterest' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the 'observables' input to the forecast method of the nb_model_generic forecast, when the 'output' input is set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **graphCorrelation** ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the ac1 output from SIM or THEO, the fourth input can be the ac1 output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_singleEq.simulatedMoments](#), [nb_singleEq.theoreticalMoments](#)
[nb_singleEq.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_estimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **help ↑**

```
helpText = nb_singleEq.help  
helpText = nb_singleEq.help(option)  
helpText = nb_singleEq.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize ↑**

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► interpretForecast ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:**Input:**

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given,

otherwise this option does not apply.

- 'optimizer'
 - : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset'
 - : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel'
 - : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters'
 - : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsg.
- 'periods'
 - : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate'
 - : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest'
 - : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch'
 - : If the 'model' input is given these are the variables that are matched, otherwise these are taken from the 'fcstDB' option.
- 'scale'
 - : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights'
 - : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_singleeq.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf** ↑

```
[irfs,irfsBands,plotter] = irf(obj,varargin)
```

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgc objects.

- 'compare' : Give true if you have a scalar nb_dsgc model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.

- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

```

- 'draws' : Number of draws for calculating girf. Default is
           1000. For Markov switching models this will set the
           number of simulated paths of states. For more see
           the 'type' input.

- 'factor' : A cell array with the factors to multiply the
             irf of the individual model variables.
             I.e. {'var1',100,...} or
                  {'var1','var2'},100,...}

If the string starts with a asterisk you will
multiply all the variables that contain that
string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This option sets the error band percentiles of the
               graph, when the 'perc' input is empty. As a 1 x
               numErrorBand double. E.g. [0.3,0.5,0.7,0.9].
               Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element
                  must be on the same format as obj.solution. I.e.
                  each element must be the solution of the model
                  given a set of drawn parameters. See the
                  parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from
                  this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

    > 'cumulative product' : cumprod(1 + X,1)

    > 'cumulative sum' : cumsum(X,1)

    > 'cumulative product (log)' : log(cumprod(1 + X,1))

    > 'cumulative sum (exponential)' : exp(cumsum(X,1))

    > 'cumulative product (%)' : cumprod(1 + X/100,1)

    > 'cumulative sum (%)' : cumsum(X/100,1)

    > 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))

    > 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))

    > '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
             Default is ''. See help on the method input of the
             nb_model_generic.parameterDraws method for more
             this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
                    the map from structural shocks to dependent

```

variables. See `nb_var.set_identification`. Of course this option only makes sense for VARs identified with sign restrictions.

- `'normalize'` : A 1 x 3 cell. First element must be the variable name as a string. The second element must be the value to normalize to, as an integer. While the third element is the period to be normalized, if set to `inf`, it will normalize the max impact period.
E.g. `{'Var',1,2}, {'Var',1,inf}`
Caution: 2 means observation 2 of the IRF, and as the IRF start at period 0, this means that observation 2 is period 1.
- `'normalizeTo'` : `'draws'` or `'mean'`. `'draws'` normalize each draw of irfs, while `'mean'` normalize the mean and scale percentiles/other draws accordingly. Default is `'draws'`.
Caution: Setting this to `'mean'` will not work for reported variables that is not measured as % deviation from steady-state/mean.
- `'newReplic'` : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the replic input.
- `'parallel'` : Give true if you want to do the irfs in parallel. This option will parallelize over models. Default is false.
- `'parallelL'` : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if `numel(obj) == 1`. Default is false.
- `'pause'` : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- `'perc'` : Error band percentiles. As a 1 x `numErrorBand` double. The input must be given as the coverage, and not the percentiles itself. E.g. `[0.3,0.5,0.7,0.9]`. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So `[0.3,0.5,0.7,0.9]` will get the percentiles `[5,15,25,35,65,75,85,95]`, i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the `'fanPerc'` input (0.68).)
- `'periods'` : Number of periods of the impulse responses.
- `'plotSS'` : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for `nb_dsg` objects.

```

- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the
initial steady state. Only an option if 'plotSS'
is set to true. Only for nb_dsg objects.

- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the
method 'identDraws'. See the option
'draws' of the method
nb_var.set_identification for more.

- 'settings' : Extra graphs settings given to nb_plots
function. Must be a cell.

- 'shocks' : Which shocks to create impulse responses of.
Default is the residuals defined by the first model.

For Markov switching or break point models it may
be wanted to only run a IRF when switching the
state. To do so set this option to {'states'}.
You should also set 'startingValues' to
'steadystate(r)', where r is the regime you start
in. For a Markov switching model also set
'startingProb' to the same r. This is only an
option if 'type' is set to 'irf'.

- 'sign' : Sign of the impulse. Either 1 or -1. Can also be
a vector of the same size as 'shocks' input.

- 'stabilityTest' : Give true if you want to discard the draws
that give rise to an unstable model. Default
is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar : Select the the regime to take the
initial transition probabilities
from.

    > double : A draws x nRegime or a 1 x
nRegime double. draws refer to
the number set by the 'draws'
input.

    > 'ergodic' : Start from the ergodic transition
probabilities. Default for 'irf'.

    > 'random' : Randomize the starting values
using the simulated starting
points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing

```

with a MS-model or a break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. For MS - models the default is the ergodic mean (default as long as 'girf' is not selected as 'type'), while for break-point models it is to start from the first regime.

```

> 'zero'          : Start from zero on all variables.

> 'random'        : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'         : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'           : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'   : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

- 'variables'       : The variables to create impulse responses of.
                     Default is the dependent variables defined by the
                     first model.

Caution: The variables reported by the reporting
          property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level
                     using the method specified by 'levelMethod'.

```

Output:

```

- irfs      : The (central) impulse response function stored in a
               structure of nb_ts object. Each field stores the the
               impulse responses of each shock. Where the variables
               of the nb_ts object is the impulse responses of the
               wanted endogenous variables.

               I.e. the impuls responses to the shock 'E_X_NW' is
               stored in irfs.('E_X_NW'). Which will then be a nb_ts
               object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the
          responses from each model are saved as
          different datasets (pages) of the nb_ts object.

```

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_singleEq.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast ↑**

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsg object is filtered or not.

Input:

- obj : An object of class nb_dsg.
- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMS** ↑

```
ret = isMS(obj)
```

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

```
ret = isMixedFrequency(obj)
```

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isNB** ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStateSpaceModel** ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

ret = isStatic(obj)

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

ret = isbayesian(obj)

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isestimated ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isforecasted ↑

```
ret = isforecasted(obj)
```

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved** ↑

```
ret = issolved(obj)
```

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **jointPredictionBands** ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint

```
prediction bands for  
macroeconomic risk management  
  
> 'copula' : Using a copula likelihood approach.  
  
> 'mostlik' : Group the paths based on how  
likely they are.  
  
- 'nSteps' : Number of forecasting steps to use when constructing the  
error bands. If empty (default) all forecasting steps stored  
in the object is used.
```

Output:

- JPB : An nb_ts object storing the joint prediction bands.
The percentiles are stored as datasets. See the
dataNames property for which page represent which
percentile.

The data of the nb_ts object has size nSteps x nVars
x nPerc.
- plotter : A nb_graph_ts object. Use the graphSubPlots method
to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior ↑**

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend
on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_singleEq.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. `@(x)gt(x,0), @(x)gt(x,0)||lt(x,1)`, i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. `@(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1)`, i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,`@(x)gt(x,0.1)`}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,`@(x)lt(x,0.1)`}, to

test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.

> 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.

> 'ss' : A N x 2 cell, where the elements of each row are:

1. The expression using any of the endogenous variables of the model.
2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_singleEq.monteCarloFiltering](#), [nb_singleeq.irf](#)
[nb_singleEq.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **mincerZarnowitzTest** ↑

`[test,pval,res] = mincerZarnowitzTest(obj,precision)`

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class `nb_model_generic`. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '`%8.6f`'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_singleEq.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **monteCarloFiltering** ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.

- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

[function_handle](#), [nb_singleEq.mcfRestriction](#), [nb_singleeq.solve](#)
[nb_singleEq.assignParameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass [NB_MODEL_GENERIC](#)

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.
 - > 'wildBootstrap' : Create artificial data by wild bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'blockBootstrap' : Create artificial data by non-overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'mblockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'rblockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" parameters based on estimation on these data. Only an option for models estimated with classical methods.
 - > 'copulaBootstrap' : Uses a copula approach to draw residual that are autocorrelated, Does not handle heteroscedasticity. Only an option for models estimated with classical methods.
 - > 'posterior' : Posterior draws. Only for models estimated with bayesian methods.

Default for models estimated with
bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more on this option.
- stable : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores available.
- 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.
- 'initialDraws' : When drawing parameters this factor decides how many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
 - > 'param' : Default. A struct with the following fields:
 - beta : A nPar x nEq x draws double with the estimated parameters.
 - sigma : A nEq x nEq x draws double with the estimated covariance matrix.
- For factor models these fields are also returned:
- lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 - R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
- factors : Draws of the factors as a T x nFac x

```

draws double.

> 'solution' : Get the solution of the model for each draw from the
distribution of parameters. The output will be a struct
with size 1 x draws. The struct will have the same
format as the solution property of the underlying
object. I.e. only one output!

> 'object'   : Get the draws as a 1 x draws nb_model_generic object.
Each element will be a model representing a given draw
from the distribution of the parameters. I.e. only one
output!

```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + ( +/- nb_distribution.t_icdf(alpha/2)) * stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```

plotter          = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
                                         increment,varargin)

```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.
 - > 'hairyplot' : Plot the recursive point forecast in the same graph as the actual data.
- startDate : A string with the start date of the forecast to plot. As a string or an object of class nb_date. Only an option when type is 'default'. Default is to plot the latest forecast produced.
- increment : Number of periods between the hairs when doing a hairy-plot. Default is 1.

Optional input:

- 'type' : If the history is of some variables are filtered, you can use this input to choose to plot the smoothed or updated estimates of the filtered variables. Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- plotter : An object of class nb_graph. Use the graphSubPlots method or the nb_graphSubPlotGUI class.
- plotFunction2Use : A string with the name of the method to call on the plotter object to produce the graphs.

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth SÃ¶therhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity ↑**

plotter = plotForecastDensity(obj,date,variable)

Description:

Plot the density forecast at a given date for a given variable. It will be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the provided date. See the dataNames property of the DB property of the returned plotter object. If the model has produced nowcast the names will 'horizon0' (a variable lag one period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► plotMCF ↑

```
plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...  
upperBound,method)
```

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot.

'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- `plotter` : > 'allInOne' : A `nb_graph_cs` object. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
 > 'biplot' : A vector of `nb_graph_data` objects with size equal to the number of pairwise combination of the parameters that can be made. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_singleEq.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFDistTest ↑**

`plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)`

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `test` : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_singleEq.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,valueName)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- `paramD` : A draws x N double with the parameter draws. E.g. use `paramD(success,:)` from the output of the `monteCarloFiltering` method. N is the number of parameters.
- `params` : A 1 x N cellstr with the names of the parameters.
- `values` : A draws x 1 double with the values of the monte carlo filtering.
- `nameValue` : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- `plotter` : A vector of `nb_graph_data` objects with size 1 x N. Use the `graph` method or the `nb_graphMultiGUI` class to produce the graphs.

See also:

[nb_singleEq.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **plotPosteriors** ↑

```
plotter = plotPosteriors(obj)
plotter = plotPosteriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).
- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.

Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_singleEq.getPosteriorDistributions](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

printed = print(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results ↑**

```
printed = print_estimation_results(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_singleeq.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **removeObservations ↑**

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties
of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **shock_decomposition ↑**

[decomp, decompBand, plotter] = shock_decomposition(obj, varargin)

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : A vector of nb_model_generic objects

- 'packages' : Cell matrix with groups of shocks and
the names of the groups. If you have not
listed a shock it will be grouped in a
group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';  
'shock_name_1' , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with
a shock name or a cellstr array with the
shock names. E.g 'E_X_NW' or {'E_X_NW', 'E_Y_NW'}.

Caution: Two special identifiers can be used;
'Initial Conditions' and 'Steady-state'. The
first represent the impact of all shocks that
hit the system before the decomposition/
estimation started. The second summarizes the
impact of steady-state changes on the system,

i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a $1 \times \text{numErrorBand}$ double. E.g. [0.3, 0.5, 0.7, 0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.

- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string. If different models has different frequency this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_singleEq.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be

a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given variable.
- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the

simulation of.

- > 'endo' : All the endogenous variables are returned.
- > 'fullendo' : All the endogenous variables are returned included the lag variables.
- > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
- > 'all' : All variables are returned (but not the lags). Also including exogenous and shocks.

- 'parallel' : Give true if you want to do the simulations in parallel. I.e. spread models to different threads. Default is false.
- 'parameterDraws' : Number of draws of the parameters. When set to 1 it will discard parameter uncertainty. Default is 1.
- 'regime' : Select the regime you want to simulate. If empty the simulation will switch between regimes. Only an option for Markov switching models.
- 'regimeDraws' : Number of drawn regime paths when doing simulations. This will have no effect if the states input is providing the full path of regimes. Default is 1.
- 'seed' : Set simulation seed. Default is 2.0719e+05.
- 'startDate' : Give the start date of the simulation. If not provided the output will be a nb_data object without a spesific start date (default). If provided the output will be an object of class nb_ts.

Caution : If dealing with break-points or exogenous time-varying parameters this input must be provided.

- 'startingValues' : Either a double or a char:

- > double : A 1 x nVar double.
- > 'mean' : Start from the mean of the data observations. Default for all model except nb_dsge models.
- > 'steadyState' : Start from the steady state. For now only an option for nb_dsge models. If you are dealing with a MS-model you can indicate the state by 'steadyState(state)'. E.g. to start in state 1 give; 'steadyState(1)'. Default for nb_dsge models.

```

> 'zero'           : Start from zero on all variables
                      (Except for the deterministic
                      exogenous variables)

- 'stabilityTest' : Give true if you want to discard the parameter draws
                     that give rise to an unstable model. Default
                     is false.

- 'startingProb'  : Either a double or a char:

    > double        : A nPeriods x nRegime or a 1 x
                      nRegime double. nPeriods refer to
                      the number of recursive periods to
                      forecast.

    > 'ergodic'     : Start from the ergodic transition
                      probabilities. Default.

```

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity ↑**

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **simulatedMoments** ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- `obj` : An object of class nb_model_generic.
- Optional inputs;
- `'output'` : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- `'stacked'` : If the output should be stacked in one matrix or not. true or false. Default is false.
- `'nLags'` : Number of lags to compute when 'stacked' is set to true.
- `'vars'` : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- `'type'` : Either 'covariance' or 'correlation'. Default is 'correlation'.
- `'draws'` : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- `'pDraws'` : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- `'perc'` : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- `'nSteps'` : Number of simulation steps. As a 1x1 double. Default is 100.
- `'burn'` : The number of periods to remove at start of the simulations. This is to randomize the starting

values of the simulation. Default is 0.

- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Default is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
 - varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
 - varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.
- E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
`getVariable(varargin{i}, 'y', 'x')`,
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use `getVariable(varargin{i}, 'x', 'y')`. (This example only works if the output is of class nb_cs)

See also:

[nb_singleEq.graphCorrelation](#), [nb_singleEq.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

`obj = solve(obj)`

Description:

Solve estimated model(s) represented by nb_singleEq object(s).

Input:

- obj : A vector of nb_singleEq objects.

Output:

- obj : A vector of nb_singleEq objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth Sæterhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_singleEq.solveNormal(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveOLSEq** ↑

```
tempSol = nb_singleEq.solveOLSEq(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_singleEq.solveRecursive(results,opt,ident)
```

Written by Kenneth Sæterhagen Paulsen

► **solveStepAhead** ↑

```
tempSol = nb_singleEq.solveStepAhead(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveTSLSeq** ↑

```
tempSol = nb_singleEq.solveTSLSeq(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveTSLSeqRecursiv** ↑

```
tempSol = nb_singleEq.solveTSLSeqRecursiv(results,opt)
```

Written by Kenneth Sæterhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **struct** ↑

```
s = struct(obj)
```

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_singleEq.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

```
options = nb_singleEq.template()
options = nb_singleEq.template(num)
```

Description:

Construct a struct which can be provided to the nb_singleEq class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

Written by Kenneth Sæterhagen Paulsen

► **testForecastRestrictions** ↑

```
probability = testForecastRestrictions(model, restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

```

- obj : A nb_model_forecast object with density forecast

- restrictions : Restrictions as a n x 3 cell array,
    {variable, date, test}.

    > variable : The variable(s) to test,
        as a string or cell array of strings.

    > date : The date as a string. If a cell array is given, a
        copy of the restriction will be made for every given
        date.

    > test : Restriction test as a function handle. The value(s)
        of the variable(s) in 'variable' at time 'date' are
        passed as arguments. Should return a logical.

```

Output:

- out : A double with the probability

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
    {'QSA_URR', 'QSA_DPO_CP'}, '2014Q1', @(x, y) x < y};
probability = model.testForecastRestrictions(restrictions);

```

See also nb_testForecastRestrictions

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

```

- obj : A scalar solved nb_model_generic object.

- expression : A MATLAB expression as a string. Use parameter names as
    variables. See model.parameters.name.

```

```

- method      : A string with the method to use. For bootstrap method see
               nb_bootstrap. For bayesian models 'posterior' is the only
               option. Default is 'bootstrap' for classical models, while
               'posterior' is the default for bayesian models.

- draws       : Number of draws from the parameter distribution. Default
               is 1000.

- alpha       : Confidence level. Default is 0.05.

- type        : Type of test:
               > '=' : Two sided test. expression == 0 (default)
               For two-sided tests it is assumed that the
               distribution is symmetric! Use
               confidenc/probabillty intervals instead.
               > '>' : One sided test. expression > 0
               > '<' : One sided test. expression < 0

```

Output:

```

- pval        : > 'classical' : P-value of test.
               > 'bayesian'  : Probabilty of test.

               A 1x1 double.

- ci          : A 1 x 2 double with the lower and upper bound of the
               confidence/probabillty interval of the tested expression.

- dist        : A nb_distribution object storing the distribution of the
               tested expression.

```

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► **theoreticalMoments** ↑

```

[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)

```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation,
autocovariance/autocorrelation.

Caution : For recursivly estimated model only the full sample estimation
results is used.

Caution : Only supported for models that can be solved in the following
way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.
- Optional inputs;
- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.
Can also be a cellstr, with a subset of variables from 'full'.
- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.
- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the all draws.

Caution: 'pDraws' must be set to a number > 1.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargout{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargout{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_singleEq.graphCorrelation](#), [nb_singleEq.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition** ↑

[data,plotter] = uncertaintyDecomposition(obj,varargin)

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the

percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).

> 'fev'	: Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc'	: At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages'	: Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1', ..., 'shock_group_name_N';
 'shock_name_1'      , ..., 'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

```
obj = uncondForecast(obj,nSteps,varargin)
```

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_singleeq.forecast](#)

Written by Kenneth Åstterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **unstruct** ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_singleeq struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update ↑](#)

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type :
 - > '' : Only update data. Default
 - > 'estimate' : Update data and estimate.
 - > 'solve' : Update data, estimate and solve
 - > 'forecast' : Update data, estimate, solve and forecast.
- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► variance_decomposition ↑

```
[decomp,decompBands,plotter,plotterBands] = ...
    variance_decomposition(obj,varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8,inf].
- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.
Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidence/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
> 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent

variables. See nb_var.set_identification. Of course this option only makes sense for VARs identified with sign restrictions.

- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1' ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output :

- decomp : A structure of nb_ts objects with the variance decomposition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomposition for each model. For each percentiles the output is as decomp.

- `plotter` : A $1 \times nModel$ vector of `nb_graph_cs` objects. Use the `graph` method or the `nb_graphPagesGUI` class to produce the graphs.
- `plotterBands`: A $1 \times nModel$ struct. Each field is a $1 \times nVars$ vector of `nb_graph_cs` objects. Use the `graphSubPlots` method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_singleEq.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

■ `nb_var`

Go to: [Properties](#) | [Methods](#)

A class for estimation and identification of VAR models.

OLS estimated VAR models uses the package `nb_olsEstimator`, while if you select the maximum likelihood estimator the `nb_mlEstimator` package is used.

The bayesian VARs are estimated using code that are based on code from a paper by Koop and Korobilis (2009), Bayesian Multivariate Time Series Methods for Empirical Macroeconomics. These code are extended to handle block exogenous variables, priors for the long run, and missing observations. See the `nb_bVarEstimator` package for the implementation.

The code of the identification algorithm is based on code provided by Andrew Binning, see paper Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". See the method `nb_var.ABidentification` for the implementation. We extend Binning (2013) to also allow for magnitude restrictions.

Superclasses:

`nb_model_generic`

Constructor:

```
obj = nb_var(varargin)
```

Optional input:

- See the `set` method for more on the inputs.
(`nb_model_estimate.set`)

Output:

- `obj` : An `nb_var` object.

See also:

`nb_model_generic`, `nb_model_estimate.set`, `nb_olsEstimator.estimate`
`nb_bVarEstimator.estimate`, `nb_mlEstimator.estimate`

Written by Kenneth Sætherhagen Paulsen

Properties:

- `addAutoName`
- `block_exogenous`
- `exogenous`
- `options`
- `results`
- `addAutoNameIfLocal`
- `dependent`
- `factors`
- `parameters`
- `solution`
- `addID`
- `endogenous`
- `forecastOutput`
- `reporting`
- `transformations`
- `estOptions`
- `name`
- `residuals`
- `userData`

- **`addAutoName`** ↑

Add automatic name (first) to default name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addAutoNameIfLocal`** ↑

Add automatic name (first) to local name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addID`** ↑

Add identifier to default name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`addIDIfLocal`** ↑

Add identifier to local name in `getName`. Default is `false`.

Inherited from superclass `NB_MODEL_NAME`

- **`block_exogenous`** ↑

A struct with the fields `name`, `tex_name` and `number`. The `name` field holds a cellstr with the names of all the block exogenous variables, the `tex_name` holds the names in the tex format. The `number` field holds a double with the number of block exogenous variables. To set it use the `set` function. E.g. `obj = set(obj,'block_exogenous',{'Var1','Var2'})`; or use the `<className>.template()` method.

- **`dependent`** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the dependent variables, the tex_name holds the names in the tex format. The number field holds a double with the number of dependent variables. To set it use the set function. E.g. obj = set(obj,'dependent',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models this property stores the variables declared as endogenous in the model file, and will be the same as the endogenous property.

Inherited from superclass NB_MODEL_GENERIC

- **endogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the endogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of endogenous variables. To set it use the set function. E.g. obj = set(obj,'endogenous',{'Var1','Var2'}); or use the <className>.template() method.

Inherited from superclass NB_MODEL_GENERIC

- **estOptions** ↑

A struct with estimation options. Output from the estimator function. E.g. nb_olsEstimator.estimate.

Inherited from superclass NB_MODEL_ESTIMATE

- **exogenous** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the exogenous variables, the tex_name holds the names in the tex format. The number field holds a double with the number of exogenous variables. To set it use the set function. E.g. obj = set(obj,'exogenous',{'Var1','Var2'}); or use the <className>.template() method.

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **factors** ↑

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the factors, the tex_name holds the names in the tex format. The number field holds a double with the number of factors. To set it use the set function. E.g. obj = set(obj,'factors',{'Var1','Var2'}); or use the <className>.template() method.

- **forecastOutput** ↑

A struct with forecast output

Inherited from superclass NB_MODEL_FORECAST

- **name** [↑](#)

Name of model. If not assign a name a generic name will be provided. See the implementation of nb_model_name.createName for different subclasses.

Inherited from superclass NB_MODEL_NAME

- **options** [↑](#)

A struct storing all the options to set for the model object of interest. Use the static <className>.help method to get help on each field of the options structure.

Inherited from superclass NB_MODEL_OPTIONS

- **parameters** [↑](#)

A struct with the parameter names of the model stored at the field name, and the parameter value stored at the field value. See also the getParameters method.

Inherited from superclass NB_MODEL_GENERIC

- **reporting** [↑](#)

A N x 3 cell matrix of how to transform model variables to reported variables. First column is the reported variable name, while the second column is the expression to convert the model variable to reported variable. The last column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Caution: During reporting, 20 periods of history of the input variables are appended to the forecast.

Can also be empty.

See also: nb_ts.checkExpressions.

Inherited from superclass NB_MODELDATA

- **residuals** [↑](#)

A struct with the fields name, tex_name and number. The name field holds a cellstr with the names of all the residuals, the tex_name holds the names in the tex format. The number field holds a double with the number of residuals. Cannot be set. Model must be solved for this proeprty to be up to date!

Caution: For nb_dsg models the residuals and exogenous property stores the same variables (after the model being solved)!

Inherited from superclass NB_MODEL_GENERIC

- **results** ↑

A struct storing all the results of the estimation.

Inherited from superclass NB_MODEL_ESTIMATE

- **solution** ↑

A struct storing the companion form of the model.

$Y_t = A*Y_{t-1} + B*X_t + C*e_t, e \sim N(0, vcv)$

Observation equation (Factor models only):

$O_t = F*X_t + G*Y_t + u_t, u \sim N(0, R)$

Observation equation (ARIMA models only):

$X_t = G*Z_t + Y_t$

Fields:

- A : See the equation above. A nendo x nendo double.
- B : See the equation above. A nendo x nexo double.
- C : See the equation above. A nendo x nres double.
- endo : A 1 x nendo cellstr with the decleared endogenous variables.
- exo : A 1 x nexo cellstr with the decleared exogenous variables.
- res : A 1 x nres cellstr with the decleared residuals/shocks.
- vcv : Variance/covariance matrix of the residuals/shocks. As a nres x nres double.
- class : The name of the model class.

Factor models and ARIMA models (only):

- factors : A 1 x nfact cellstr with the estimated factors and/or exogenous variables of the measurement equation.
- F : See the equation above. A nobs x nExo double.
(Storing the impact of exogenous terms of the observation equation)
- G : See the equation above. A nobs x nfact double.
- observables : A 1 x nobs cellstr with the decleared observable variables.
- R : Variance/covariance matrix of the residuals/shocks of the observation equation.

As a nobs x nobs double. If not provided, it is assumed to be 0.

DSGE:

- ss : A nendo x 1 double with the steady-state of the model.
- jacobian : A neq x (nforward + nendo + nbackward + nres) sparse double storing the jacobian of the model.
- jacobianType: A 1 x (nforward + nendo + nbackward + nres) indicating the type of each column of the jacobian. 1 indicate leaded variables, 0 current period variables, -1 indicate lagged variables, and 2 indicate residuals (innovations).

DSGE (with break-points):

In this case the fields A, B, C and ss are all cell arrays with size 1 x nbbreaks. Each element has the size in the standard case.

Inherited from superclass NB_MODEL_GENERIC

- **transformations** ↑

A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

All methods of the nb_math_ts class that perserve the time span are supported in expression. You can also call user defined functions, as long as they take and return a nb_math_ts object. A nb_math_ts object is just a double with a time dimension! To get a list of all the methods of the nb_math_ts class use; methods('nb_math_ts').

Can also be empty.

See also: nb_ts.createVariable and nb_ts.createShift.

Inherited from superclass NB_MODELDATA

- **userData** ↑

Adds the possibility to add user data. Can be of any type

Inherited from superclass NB_MODEL_GENERIC

Methods:

- ABidentification
- appendData
- applyLongRunPriors
- assignParameters
- assignPosteriorDraws
- assignTexNames
- calculateMultipliers
- calculateStandardError
- checkModel
- checkPosteriors
- checkReporting
- conditionalTheoreticalMoments
- constructCondDB
- constructScore
- convert
- convertEach
- createVariables
- dieboldMarianoTest
- doForecastPerc2Dist
- doForecastPerc2ParamDist
- doSimulateFromDensity
- empiricalMoments
- eq
- estimate
- evalFcstAtDates
- evaluateForecast
- evaluatePrior
- forecast
- forecastPerc2Dist
- forecastPerc2ParamDist
- getActual
- getCondDB
- getDSGEVARPriorMoments
- getDependent
- getDependentNames
- getEstimationOptions
- getEstimationStartDate
- getFiltered
- getForecast
- getForecast Variables
- getHistory
- getIdentification
- getIdentifiedResidual
- getLHSVars
- getModelNames
- getModelVars
- getOriginalVariables
- getPIT
- getParameterDrawsMethods
- getParameters
- getPosteriorDistributions
- getPredicted
- getPriorDistributions
- getRecursiveEstimationGraph

- `getRecursiveScore`
- `getResidualGraph`
- `getRoots`
- `getSolution`
- `grangerCausalityTest`
- `handleCondInfo`
- `help`
- `initialize`
- `irf`
- `isFiltered`
- `isMS`
- `isNB`
- `isStatic`
- `isestimated`
- `issolved`
- `loadPosterior`
- `mincerZarnowitzTest`
- `parameterDraws`
- `plotForecast`
- `plotMCF`
- `plotMCFValues`
- `plotPriors`
- `print Cov`
- `priorTemplate`
- `set`
- `setPrior`
- `shock_decomposition`
- `getResidual`
- `getResidualNames`
- `getScore`
- `getVariablesList`
- `graphCorrelation`
- `handleMissing`
- `identDraws2Solutions`
- `interpretForecast`
- `isDensityForecast`
- `isIdentified`
- `isMixedFrequency`
- `isStateSpaceModel`
- `isbayesian`
- `isforecasted`
- `jointPredictionBands`
- `mcfRestriction`
- `monteCarloFiltering`
- `parameterIntervals`
- `plotForecastDensity`
- `plotMCFDistTest`
- `plotPosteriors`
- `print`
- `print_estimation_results`
- `removeObservations`
- `setMeasurementEqRestriction`
- `set_identification`
- `simulate`

- simulateFromDensity
 - solve
 - solveRecursive
 - stateSpace
 - template
 - testParameters
 - uncertaintyDecomposition
 - unstruct
 - variance_decomposition
 - simulatedMoments
 - solveNormal
 - solveVector
 - struct
 - testForecastRestrictions
 - theoreticalMoments
 - uncondForecast
 - update
-

► ABidentification ↑

```
S = nb_var.ABidentification(ident,A,sigma,maxDraws,draws,ask)
```

Description:

Identification of VAR model using combination of zero and sign restrictions. See Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". Many thanks to him for supplying help!

As a extension to Binning (2013) this code also allow for magnitude restrictions.

Input:

- ident : The identification assumption, as a struct. See set_identification method of the nb_var class for more
- A : Transition matrix
- sigma : Covariance matrix of the VAR residuals.
- maxDraws : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.
- draws : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Output:

- S : A nb_struct with field W, with the stored map from structural shocks to dependent variables.

Written by Kenneth Sæterhagen Paulsen,
Subfunctions of this code are written by Andrew Binning.

► **appendData** ↑

```
obj = appendData(obj,DB)
```

Description:

Append more data to existing data stored in the nb_model_generic object.

Caution : If append == 1 (true) then the DB input has precedence over the existing dat of the object.

Input:

- obj : An object of class nb_model_generic
- DB : An object of class nb_ts or nb_cs

Optional input:

- interpolateDate : See nb_ts.merge for more help on this input.
- method : See nb_ts.merge for more help on this input.
- rename : See nb_ts.merge for more help on this input.
- append : See nb_ts.merge for more help on this input.
- type : See nb_ts.merge for more help on this input.

Output:

- obj : An object of class nb_model_generic

See also:

[nb_ts](#), [nb_cs](#), [nb_ts.merge](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **applyLongRunPriors** ↑

```
obj = applyLongRunPriors(obj,H,phi)
```

Description:

Apply priors for the long run as in Giannone et. al (2014).

If the prior field of the options property is not yet assign a prior, a jeffrey prior is used as default.

Caution: Please do not call this method before nb_var.setPrior!

Input:

- obj : A N x M matrix of nb_var objects, with nDep dependent variables.

- phi : Either a nDep x 1 double vector with the prior shrinkage parameter for each equation, or a m x 2 cell. m <= nDep. The format of the cell input must be as follows;

```
{'var1',1;'var2',0.5}
```

If some dependent variables are not assigned any parameter the default is 1.

- H : A q x nDep matrix, where q <= nDep. Each row is the cointegration vector to apply on the prior. If q < nDep the null space will be applied to the rest of the rows.

Or it can be a q x 1 cellstr with the cointegration relations. Again q <= nDep, and if q < nDep the null space will be applied to the rest of the rows. Please note that only dependent variables may be applied in the cointegration relations! Example:

```
{'var1 - var2','var3 + var1'}
```

Output:

- obj : A N x M matrix of nb_var objects.

See also:

[nb_var.set](#), [nb_var.setPrior](#)

Written by Kenneth Å!terhagen Paulsen

► assignParameters ↑

```
obj = assignParameters(obj,varargin)
```

Description:

Assign calibrated parameters of underlying model.

Input:

Either:

> One input:

- A struct with the fieldnames as the parameter names and the fields as the parameter values.

> Two input pairs:

```

- 'param' : The names of the assign parameters. As a cellstr vector.

- 'value' : The values of the assign parameters. As a double vector.

> Optional number of input pairs:

- 'beta'   : Assign all the estimated parameters of the main equation.
              Must be of same size as obj.results.beta.

- 'sigma'  : Assign all the estimated parameters of the covariance
              matrix of the main equation. Must be of same size as
              obj.results.sigma.

- 'lambda' : Assign all the estimated parameters of the observation
              equation. Must be of same size as obj.results.lambda. Only
              for factor models.

- 'R'      : Assign all the estimated parameters of the covariance
              matrix of the observation equation. Must be of same size
              as obj.results.R. Only for factor models.

```

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignPosteriorDraws** ↑

obj = assignPosteriorDraws(obj, varargin)

Description:

Assign posterior draws made outside the NB Toolbox.

Input:

Either

> Two input pairs:

- 'param' : The names of the assign parameters. As a cellstr vector.
- 'value' : The values of the assign parameters. As a double nParam x nReg x nDraws vector.

> Optional number of input pairs:

- 'beta' : Assign all the posterior draws of the parameters of the main equation. Must be of same size as obj.results.beta. The draws must come in the third dimension.

- 'sigma' : Assign all the posterior draws of the parameters of the covariance matrix of the main equation. Must be of same size as obj.results.sigma. The draws must come in the third dimension.
- 'lambda' : Assign all the posterior draws of the parameters of the observation equation. Must be of same size as obj.results.lambda. Only for factor models. The draws must come in the third dimension.
- 'R' : Assign all the posterior draws of the parameters of the covariance matrix of the observation equation. Must be of same size as obj.results.R. Only for factor models. The draws must come in the third dimension.
- 'period' : When the model is recursively estimated, you can choose which period your current posterior draws are to be assigned to. Default is the last period.

Output:

- obj : An object of class nb_model_generic, where the parameters of the underlying model has been changed. See solve to get the solution of the model with the updated parameters.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **assignTexNames ↑**

obj = assignTexNames(obj, names, texNames)

Description:

Assign tex names to variables and parameters. Tex names should not include subscripts for any variable, as _{t} is automatically appended to each variable! Use superscripts instead.

Input:

- obj : An object of class nb_model_generic.
- names : A Nx1 cellstr with the names of the variables and/or parameters of the model.
- texNames : A Nx1 cellstr with the corresponding tex names of the variables and/or parameters of the model.

Output:

- obj : A nb_model_generic object where the dependent, endogenous, exogenous, residuals or parameters properties has been set some of its tex_name fields. (Also observables and observablesFast for factor model.)

See also:

[nb_dsge.writeTex](#), [nb_dsge.writePDF](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateMultipliers** ↑

```
mult = calculateMultipliers(obj,varargin)
```

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Caution: This function assumes that the variables given to the variables input is delta_Y and delta_G not Y and G of the formula (3)!

Input:

- obj : An object of class nb_dsge.

Optional input:

- 'variables' : A cellstr with the variables to produce the multipliers of. Must be provided!
- 'instrument' : A one line char with the name of the instrument. Must be provided!
- 'rate' : A one line char with the name of the interest rate to use calculate the net present value. Must be provided!
- 'shocks' : A cellstr with the shocks to produce the multipliers of. Must be provided!
- 'gross' : Give false if the rate input is the net interest rate. Default is true, i.e. that the rate input is the gross interest rate.
- 'perc' : Error band percentiles. As a 1 x nPerc double. E.g. [0.3,0.5,0.7,0.9]. If empty all the simulation will be returned.
- rest : Use the 'replic' option to produce error bands.

Output:

- mult : A nShocks x nVars x nSim (nPerc) nb_cs object. Use the double method on this output to convert it to a double matrix.

See also:

[nb_var.irf](#), [nb_calcMultiplier](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **calculateStandardError** ↑

obj = calculateStandardError(obj,method,draws)

Description:

Calculate standard errors by bootstrap (classical) or posterior draws (bayesian).

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Input:

- obj : A scalar nb_model_generic object
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.

Output:

- obj : A scalar nb_model_generic object, where the results property is updated with the bootstrapped standard errors.
Extra outputs (all based on bootstrapped draws):
 - stdBeta : Standard deviation of coefficient of the main equation.
 - tStatBeta : T-statistic for the coefficient of the main equation.
 - pValBeta : Calculated P-value based on a kernel density estimator of the parameter distribution. Two-sided test against the null hypothesis of the parameters being 0.
 - stdLambda : Standard deviation of coefficient of the observation equation.

```
- tStatLambda : T-statistic for the coefficient of the  
observation equation.  
  
- pValLambda : Calculated P-value based on a kernel  
density estimator of the parameter  
distribution. Two-sided test against the  
null hypothesis of the parameters being 0.  
  
- ci : Confidence interval/probability interval calculated based on  
a kernel density estimation instead of assuming asymptotic  
normal distribution. Only for the parameter in beta! As a  
(nCoeff * nEq) + 1 x 3 cell array.  
  
- dist : Estimated distribution of the coefficients of the main  
equation. As a (nCoeff * nEq) x 1 nb_distribution object.
```

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkModel** ↑

```
obj = checkModel(obj)
```

Description:

Secure that the object is up to date when it comes to the options,
estOptions and results struct properties.

Input:

```
- obj : An object of class nb_model_generic.
```

Output:

```
- obj : An object of class nb_model_generic.
```

Written by Kenneth Återhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **checkPosteriors** ↑

```
[DB,plotter,pAutocorr] = checkPosteriors(obj,varargin)
```

Description:

Plot posteriors draws in the order they were drawn. This to check for
problems with posterior draws that may be autocorrelated.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optional input:

- 'nLags' : Number of lags to include in the autocorrelation plot. Default is 10.
- 'iter' : The recursive iteration date. Either as a string or a nb_date object. If recursive estimation is done, this input must be provided.

Output:

- DB : A nb_data object with size nDraws x numCoeff.
- plotter : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.
- pAutocorr : A nb_graph_data object. Use the graphSubPlots or the nb_graphSubPlotGUI class to plot it on screen.

Caution: Only plots the autocorrelation for the first chain if the sampler has used more than one chain.

See also:

[nb_graph_data](#), [nb_graphSubPlotGUI](#), [nb_model_sampling.checkUpdatedPriors](#)

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► checkReporting ↑

obj = checkReporting(obj)

Description:

Check reporting, and store historical observation of reported variables.

Input:

- obj : A NxM nb_modelData object with the reporting property set.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODELDATA

► conditionalTheoreticalMoments ↑

```
c = conditionalTheoreticalMoments(obj,varargin)
```

Description:

Calculate conditional theoretical moments; I.e.
(auto)covariance/(auto)correlation matrix.

Caution : This method stack all autocorrelations in one matrix.
Equivalent to the output of the theoreticalMoments method
when the 'stacked' input is set to true.

Caution : For recursively estimated model only the full sample estimation
results is used.

Input:

- obj : An object of class nb_model_generic

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return
the moment of the dependent variables of the model without
lag, while full will include all lags as well. 'dependent'
is default.

Can also be a cellstr, with a subset of variables
from 'full'.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'type' : Either 'covariance' or 'correlation'. Default is
'correlation'.

- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the
estimate parameters.

- 'perc' : Error band percentiles. As a 1 x numErrorBand double.
E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
all draws.

Caution: 'draws' must be set to a number > 1.

- 'method' : The selected method to make the parameter draws.
Default is ''. See below for what that means.

> 'bootstrap' : Create artificial data to bootstrap
the estimated parameters. Default
for models estimated with classical
methods.

> 'wildBootstrap' : Create artificial data by wild
bootstrap, and then "draw" parameters
based on estimation on these data.
Only an option for models estimated
with classical methods.

> 'blockBootstrap' : Create artificial data by
non-overlapping block bootstrap,

and then "draw" paramters based on estimation on these data.
 Only an option for models estimated with classical methods.

> 'mBlockBootstrap' : Create artificial data by overlapping block bootstrap, and then "draw" paramters based on estimation on these data.
 Only an option for models estimated with classical methods.

> 'rBlockBootstrap' : Create artificial data by overlapping random block length bootstrap, and then "draw" paramters based on estimation on these data.
 Only an option for models estimated with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild overlapping random block length bootstrap., and then "draw" paramters based on estimation on these data.
 Only an option for models estimated with classical methods.

> 'posterior' : Posterior draws. Only for models estimated with bayesian methods.
 Default for models estimated with bayesian methods.

Caution : Posterior draws is already made at the estimation stage.

- 'nSteps' : Number of steps ahead. As an integer.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **constructCondDB** ↑

```
condDB          = nb_model_generic.constructCondDB(data,exo)
[condDB,shockProps] = nb_model_generic.constructCondDB(data,exo,endo,...  

                                         horizon,shocks,activeShocks,shockHorizon)
```

Description:

Restrict conditional information.

Static method.

Input:

- data : An nb_ts object.
- exo : The exogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- endo : The endogenous variables to condition on. As a cellstr. Must be included as a variable of the data input.
- horizon : The number of periods to condition on the endo variables. Either as a scalar or a vector of size 1 x nEndo. Default is to use the number of observations of the data input.
- shocks : The shocks to condition on. As a cellstr. Must be included as a variable of the data input.
- activeShocks : A cellstr with the shocks to match your endogenous restrictions. Size 1 x nActiveShock.
- shockHorizon : The number of periods to activate the given shock. Either as a scalar or a vector of size 1 x nActiveShock. Be aware that you must at least have as many periods of active shocks as you have endogenous restrictions. (It may also be important which shock(s) to meet your restrictions!).

Caution: This option will overrun the restrictions you make with the shocks input when you call the forecast method!

Output:

- condDB : The restricted conditional database, which can be given as input to the forecast method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► constructScore ↑

```
score = nb_model_generic.constructScore(forecastOutput,type)
score = nb_model_generic.constructScore(forecastOutput,type, ...
                                         allPeriods,startDate,endDate,nSteps,rollingWindow,lambda)
```

Description:

Construct forecast evaluation score given forecast output.

Input:

- forecastOutput : See the forecastOutput property of the nb_model_forecast class
- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error.
 - 'MSE' : One over mean squared error.
 - 'MAE' : One over mean absolute error.
 - 'ME' : Mean error.
 - 'STD' : One over Standard error of the forecast error.
 - 'ESLS' : Exponential of the sum of the log scores.
 - 'EELS' : Exponential of the mean of the log scores.
 - 'MLS' : Mean log score.
- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. False is default.
- startDate : The date to begin constructing the score. Can be empty.
- endDate : The date to end constructing the score. Can be empty.
- nSteps : Select the number of forecasting steps. Can be empty.
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- score : As a nHor x nVar x nPeriods

See also:

[nb_model_forecast.getScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **convert ↑**

```
obj = convert(obj,freq,method,varargin)
```

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convert.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convert.

See also:

[nb_ts.convert](#)

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **convertEach** ↑

obj = convert(obj,freq,methods,varargin)

Description:

Convert the frequency of the data of the nb_modelData object. Both the data property will be transformed to the wanted frequency.

Caution: The shift property is not handle correctly, so de-trending cannot be used in this case!

Input:

- See the documentation of nb_ts.convertEach.
- 'options' : Provide this as one of the optional inputs, and the conversion of the frequency will be applied to options.data and not the dataOrig property.

Output:

- See the documentation of nb_ts.convertEach.

See also:

nb_ts.convert

Written by Kenneth S. Paulsen

Inherited from superclass NB_MODELDATA

► **createVariables** ↑

```
obj = createVariables(obj,expressions,fcstHorizon)
[obj,plotter] = createVariables(obj,expressions,fcstHorizon)
```

Description:

Create model variables given input data.

Caution : Only for time-series models!

Input:

- obj : A NxM nb_modelData object
- expressions : A N x 4 cell matrix of how to transform input data to model variables. First column is the model variable name, while the second column is the expression to convert the input data to model variable. The third column is the expression on how to calculate the shift variable. While the fourth column is for comments.

More on:

- > second column : Any expression that can be interpreted by the nb_ts.createVariable method.
 - > third column : Any expression that can be interpreted by the nb_ts.createShift method.
 - > fourth column : Comments
- fcstHorizon : The forecast horizon. As an integer. Default is 8. It is important that this option is higher than the number of forecasting/irf steps, or else the output will be nan!

Output:

- obj : The NxM object itself added the model variables. The expressions input is stored in the property transformations.
- plotter : A NxM nb_graph_ts object with a graph with the de-trending. Use the graphInfoStruct method or the nb_graphInfoStruct class to produce the graph.

Examples:

```

expressions = {%
  Name, input, shift, description
  'VAR1_G', 'pcn(VAR1)', 'avg', 'VAR1 growth'
  'VAR2_G', 'pcn(VAR2)', 'avg', 'VAR2 growth'
  'VAR3_G', 'pcn(VAR3)', 'avg', 'VAR3 growth'};

model = model.createVariables(expressions)

```

See also:

[nb_model_generic.transformations](#), [nb_model_generic.reporting](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODELDATA

► **dieboldMarianoTest** ↑

```
[test,pval,res] = dieboldMarianoTest(obj1,obj2,varargin)
```

Description:

Diebold-Mariano test. The null is that the models produce equally good forecast.

Only the forecast variables from both models are tested! I.e. the intersect.

Caution : The actual data is taken from obj1.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj1 : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- obj2 : An object of class nb_model_generic. You need to call one of the forecasting functions first!

Optional inputs:

- 'startDate' : The start date of the test, as a string or a nb_date object. If it is not provided the default is to use the first date which there exist forecast from the two models.
- 'endDate' : The end date of the test, as a string or a nb_date object. If it is not provided the default is to use the last date which there exist forecast from the two models.
- 'precision' : The precision of the printed result. As a string. Default is '%8.6f'.

- 'bandWidth' : The selected band width of the frequency zero spectrum estimation. Default is to use a automatic selection criterion. See the 'bandWithCrit' input. Must be set to an integer greater then 0.
- 'bandWithCrit' : Band width selection criterion. Either:
 - > 'nw' : Newey-West selection method. Default.
 - > 'a' : Andrews selection method. AR(1) specification.
- 'multivariate' : Set to true to do the multivariate test, i.e. test for equal panel forecast. Default is to test each variable separately.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_var.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **doForecastPerc2Dist ↑**

obj = doForecastPerc2Dist(obj, draws)

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- obj : A nb_model_forecast object.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

`nb_var.forecast, nb_var.forecastPerc2Dist`

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass `NB_MODEL_FORECAST`

► **doForecastPerc2ParamDist ↑**

`[obj,dist] = doForecastPerc2ParamDist(obj,distribution,draws,varargin)`

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the `nb_distribution.perc2ParamDist` function.

Input:

- `obj` : A `nb_model_forecast` object.
- `distribution` : The distribution you want to estimate based on the percentiles. You must have at least as many percentiles as there are parameters!
- `draws` : Number of draws to make from the estimated distributions.

Optional input:

- `'optimizer'` : See doc of the optimizer input to the `nb_callOptimizer` function. Default is `'fmincon'`.
- `'optimset'` : See doc of the `opt` input to the `nb_callOptimizer` function.

Output:

- `obj` : A `nb_model_forecast` object. See the property `forecastOutput`.

See also:

`nb_var.forecast, nb_distribution.perc2ParamDist`

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_FORECAST`

► **doSimulateFromDensity ↑**

`obj = doSimulateFromDensity(obj,draws)`

Description:

Simulates from the estimated kernel density.

Input:

- obj : A nb_model_forecast object.
- draws : Number of draws to use for simulation.

Output:

- obj : A nb_model_forecast object. See the property forecastOutput.

See also:

[nb_model_generic.forecast](#), [nb_var.simulateFromDensity](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **empiricalMoments** ↑

```
[m,c] = empiricalMoments(obj,varargin)
[m,c,ac1] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2] = empiricalMoments(obj,varargin)
[m,c,ac1,ac2,...] = empiricalMoments(obj,varargin)
```

Description:

Calculate empirical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation of the data of the nb_model_generic object. The options.data property must contain the selected variables!

Caution: This method only works on time-series models

Input:

- obj : An object of class nb_model_generic
- Optional inputs;
- 'vars' : Either 'dependent' or a cellstr with the wanted variables. 'dependent' will return the moment of the dependent variables. 'dependent' is default.
 - 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
 - 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
 - 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'startDate' : The start date of the calculations. Default is the start date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'endDate' : The end date of the calculations. Default is the end date of the options.data property. Only for time-series! Must be a string or a nb_date object.
- 'demean' : true (demean data during estimation of the autocovariance matrix), false (do not). Default is true.

Output:

- varargout{1} : The mean, as a 1 x nVar nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : X > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use
getVariable(varargin{i}, 'y', 'x'),
while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_var.graphCorrelation](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [eq](#) ↑

ret = eq(obj,other)

Description:

Test if nb_model_generic objects are equal. In the sens they have the same properties.

Input:

- obj : A nb_model_generic object either with same size as others, or equal to 1.
- others : A nb_model_generic model with dim == 3 or less

Output:

- ret : A logical matching the others input

Examples:

```
v(1,5) = nb_var;  
ind     = v(1) == v
```

See also:

[isequal](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► estimate ↑

```
obj      = estimate(obj,varargin)  
[obj,valid] = estimate(obj,varargin)
```

Description:

Estimate the model(s) represented by nb_model_estimate object(s).

Input:

- obj : A vector of nb_model_generic objects.
- 'parallel' : Use this string as one of the optional inputs to run the estimation in parallel.
- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is given.
- 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
- 'waitbar' : Use this string to give a waitbar during estimation. I.e. when looping over models. If 'parallel' is used this option is not supported!
- 'write' : Use this option to write errors to a file instead of throwing the error.

Optional input:

- varargin : See the the set method.

Output:

- obj : A vector of nb_model_generic objects, where the estimation results are stored in the property results.
- valid : A logical with size nObj x 1. true at location ii if estimation of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_model_generic](#), [solve](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **evalFcstAtDates ↑**

```
[evaluation,errorDates] = evalFcstAtDates(obj, dates, type, density,...  
quart)
```

Description:

Evaluates a models forecasting capabilities compared to actual data. If a date is given before day number 15 it uses data from up until two months before. If a date is given on day number 15 or later, it uses data from up until one month before the date. For quarterly data it uses the same, but restricted to also beeing in the first month in the quarter, i.e. if it is the second or third month it will always use data up until one quarter before the released.

Input:

- obj : A nb_model_generic or nb_model_group object containing forecasts.
- dates : A cellstring containing the dates that the forecast should be evaluated at. Can also be a nb_date object.
- type : The evaluation criteria. A cellstr with one or more of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores

```
- 'EELS'   : Exponential of the mean of the log scores
- 'MLS'    : Mean log score

- density : Set true if you have a density forecast.

- quart   : Set true if you want to compare the quarterly forecasts. This
option automatically detects and uses the quarterly estimates
when evaluating.
```

Output:

- evaluation : A nb_data object with the evaluation scores for each observation and variable.
- errorDates : Since it is not always the case that all of the dates are being used, errorDates returns those dates as a cellstr.

Examples:

```
m = evalFcstAtDates(b,d',{'MSE'},0,1)
m = evalFcstAtDates(b,d',{'MSE','RMSE'},0,1)
```

See also:

[nb_evalForecastfromFame](#)

Written by Tobias Ingebrigtsen

Inherited from superclass NB_MODEL_FORECAST

► **evaluateForecast** ↑

```
obj = evaluateForecast(obj,varargin)
```

Description:

Evaluate forecast using (a) selected loss function(s)

Input:

- obj : An object of class nb_model_generic.

Optional Input:

- 'draws' : Number of draws used when simulating from the distribution. Default is 1000.
- 'fcstEval' : See the documentation of the same input in the forecast method.
- 'compareToRev' : See the documentation of the same input in the forecast method.
- 'compareTo' : See the documentation of the same input in the forecast method.

Output:

- obj : An object of class nb_model_generic.

See also:

[nb_model_generic.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► evaluatePrior ↑

```
logPriorD = nb_model_generic.evaluatePrior(prior,par)
```

Description:

Evaluate the prior at a point in the parameter space.

Input:

- prior : See options.prior.
- par : A nEst x 1 double with the value of the point of evaluation.

Output:

- logPriorD: The log prior density at the evaluated point.

See also:

[nb_dsge.objective](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► forecast ↑

```
obj      = forecast(obj,nSteps,varargin)
[obj,valid] = forecast(obj,nSteps,varargin)
```

Description:

Produced conditional point and density forecast of nb_model_generic objects.

The conditional forecasting routines in this code is based on work by Junior Maih. As in his code it also handle anticipated shocks (only

for foward looking models!). See Junior Maih (2010), Conditional forecast in DSGE models.

If the 'condDB' input is set to a nb_distribution object the code uses the algorithm given in the paper Kenneth S. Paulsen (2010), Conditional forecasting with DSGE models - A conditional copula approach.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional input:

- 'waitbar' : Give 'waitbar' anywhere as an optional input to create a waitbar that iterates the models being forecasted.
 - 'write' : Give 'write' anywhere as an optional input to write errors to a file instead of throwing them. This input is not stored to the forecastOutput property of the objects.
 - 'remove' : Give 'remove' to remove the models that return errors from the obj output. This input is not stored to the forecastOutput property of the objects.
 - 'fcstEval' :
 - '' : No forecast evaluation. Default
 - 'se' : Square error forecast evaluations.
 - 'abs' : Absolute forecast error
 - 'diff' : Forecast error
- Only for density forecast:
- 'logScore' : Log score
- Can also be a cellstr with the above listed evaluation types.
- Caution: If you have done recursive estimation, this option, if not empty, will do out of sample forecasting. Else it will do in sample forecasting, i.e. use the estimated parameters for the whole sample.
- 'estDensity' : The method to estimate the density. As a string. Either:
 - 'normal' : Normal density approximation.
 - 'kernel' : Kernel density estimation. Default.
 - 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!

```

- 'parallel'      : Give true if you want to do the forecast in
                    parallel. I.e. spread models to different threads.
                    Default is false.

- 'cores'         : The number of cores to open, as an integer. Default
                    is to open max number of cores available. Only an
                    option if 'parallel' is set to true.

- 'parameterDraws' : Number of draws of the parameters. When set to 1
                     it will discard parameter uncertainty. Default is 1.

- 'draws'          : Number of residual/innovation draws per parameter
                     draw. Default is 1, i.e. only produce point
                     forecast.

- 'regimeDraws'   : Number of drawn regime paths when doing density
                     forecast. This will have no effect if the states
                     input is providing the full path of regimes. Default
                     is 1.

- 'newDraws'       : When out of parameter draws, this factor decides how
                     many new draws are going to be made. Default is 0.1.
                     I.e. 1/10 of the parameterDraws input.

- 'perc'           : Error band percentiles. As a 1 x numErrorBand
                     double. The input must be given as the coverage, and
                     not the percentiles itself. E.g. [0.3,0.5,0.7,0.9].
                     0.3 means to return the two percentiles 35 and 65,
                     which will cover 30% of the distribution around the
                     median. So [0.3,0.5,0.7,0.9] will get the
                     percentiles [5,15,25,35,65,75,85,95], i.e. this
                     will restrict the percentiles to be symmetric.
                     Default is 0.9.

If set to empty, all simulations will be returned.

Caution: 'draws' must be set to produce density
forecast.

- 'method'         : The selected method to create density forecast.
                     Default is ''. See help on the method input of the
                     nb_model_generic.parameterDraws method for more
                     this input.

- 'bins'            : The length of the bins og the domain of the density.
                     Either;

                     > []      : The domain will be found. See
                     nb_getDensityPoints (default is that 1000
                     observations of the density is stored).

                     > integer : The min and max is found but the length
                     of the bins is given by the provided
                     integer.

                     > cell     : Must be on the format:

                     {'Var1',lowerLimit1,upperLimit1,binsL1;
                      'Var2',lowerLimit2,upperLimit2,binsL2;

```

```

    ...
}

- lowerLimit1 : An integer, can be nan,
  i.e. will be found.

- upperLimit1 : An integer, can be nan,
  i.e. will be found.

- binsL1      : An integer with the
  length of the bins. Can
  be nan. I.e. bins length
  will be adjusted to
  create a domain of 1000
  elements.

Caution : The variables not included will be given
the default domain. No warning will be
given if a variable is provided in the
cell but not forecast by the model. (This
because different models can forecast
different variables)

Caution : The combineForecast method of the
nb_model_group class is much faster if the
lower limit, upper limit! Or else
it must simulate new draws and do kernel
density estimation for each model again
with the shared domain of all models
densities.

- 'startDate'   :
  : The start date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is not empty this will set
the start date of the recursive forecast.
Default is to start from the first possible
date.

- 'endDate'    :
  : The end date of forecast. Must be a string or an
  object of class nb_date. If empty the estim_end_date
  + 1 will be used.

Caution: If 'fcstEval' is empty this input will
have nothing to say.

- 'saveToFile'  :
  : Logical. Save densities and domains to files. One
  file for each model. Default is false (do not).

- 'observables' :
  : A cellstr with the observables you want the
  forecast of. Only for factor models. Will be
  discarded for all other types of models.

- 'condDB'       :
  : One of the following:

    > A nb_ts object with size nSteps x nCondVar
    with the information to condition on.

  If you condition on endogenous variables (possibly

```

residuals/shocks also) this function will find the linear combination of the shock that minimize its variance.

- > A nb_data object with size nSteps x nCondVar x nPeriods with the information to condition on. The dataNames property must be set to the start dates of the recursive conditional information. nPeriods will be the number of recursive periods.
- > A nb_ts object with size nSteps x nCondVar x 3 with the information to condition on. Page 2 must include the upper bound and page 3 the lower bound of the truncated normal distribution for the given conditional information. Both bound can be given as nan (-inf or inf). Be aware that it is also possible to use truncated distribution with the next input type, but that the algorithm used by this options is much faster. (Of course here all conditional information must be either normal or truncated normal.)
- > A nb_distribution object with size nSteps x nCondVar x 1 with the distributions to condition on. The defualt is to assume that the shock are iid with mean = 0 and std = sqrt(diag(model.vcv))

If you condition on endogenous variables (possibly residuals/shocks also) this function will find the linear combination of the shock that minimize its variance for each draw from the multivariate distribution created by using a copula with the provided marginals and the correlation matrix (See inputs.sigma).

All the variables in the x_t matrix must be provided. I.e. model.exo variables. (Except seasonals, constant and time-trend). The data must be ordered accordingly to condDBVars (second dimension), and the first observation is taken to be startInd, as long as the input is not nb_ts or nb_data.

To be able to match your restrictions on endogenous variables to a specified group of shock use the shockProps input.

Caution : If empty unconditional forecast will be produced.

- 'condDBType' : 'hard' or 'soft'. When 'hard' is choosen it is assumed that all conditional information should be interpreted with no uncertainty, i.e. actual data. When 'soft' the conditional data is interpret with uncertainty and the distribution is either taken from the 'condDB' input or simulated from the model. Default is 'soft'.
- 'condDBVars' : A cellstr with the variables to condition on. Can

include both endogenous, exogenous and shock variables. Only needed when condition on a nb_distribution object.

- 'condDBStart' : Do the conditional information contain intial values or not. Set to 0 to indicate that the conditional information also contain intial values. In this case all variables that have missing observation for the initial values are replaced with the data supplied by the condDB input. Otherwise the estimated initial observations for missing data is used (default).
- 'shockProps' : A 1 x nRes struct with fields:
 - > Name : The name of the shock/residual to use to match the conditional information. If an shock/residual is not included here, it means that it is not activated.
 - > StandardDeviation : Standard deviation of the shock. This option can be used to adjust the standard deviation away from the estimated one. If given as nan the estimated one is used.
 - Caution: If a shock is included in the condDB input, and it is a nb_distribution object. The std is taken from that distribution instead.
 - If not provided. Model stds are used.
- > Horizon : Anticipated horizon of the shocks/residual. 1 is default.
- > Periods : The active periods of the shocks/residual. If nan, the shocks/residual is active for all periods. Must be given!
- 'kalmanFilter' : Set to true to use a Kalman filter to do conditional forecasting (on endogenous variables). Only supported for the nb_var and nb_pitvar classes.
- 'sigma' : A nCondVar*nSteps x nCondVar*nSteps double with the correlation matrix of the endogneous variables to condition on. Used when drawing from the copula.

This must be the stacked autocorrelation matrix of the multivariate distribution to draw the endogenous variables from. As the endogneous variables are autocorrelated all variables for all periods must be draw at the same time, using this

autocorrelation matrix.

- 'sigmaType' : Type of correlation calculated. For more on this input see nb_copula.type.
- 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.
 - > 'fullendo' : All the endogenous variables are returned included the lag variables.
 - > 'full' : All the variables are returned included the lag variables. Also including exogenous and shocks.
 - > 'all' : All variables are returned (but not the lags). Including exogenous and shocks.
- 'startingValues' : Either a double or a char:
 This option is mainly used for making simulations from a model.
 - > double : A nPeriods x nVar or a 1 x nVar double. nPeriods refer to the number of recursive periods to forecast.
 - > 'mean' : Start from the mean of the data observations. Default.
 - > 'steadystate' : Start from the steady state. For now only an option for nb_dsgc models. If you are dealing with a MS-model or break-point model you can indicate the regime by 'steadystate(regime)'. E.g. to start in regime 1 give; 'steadystate(1)'. If the regime is not given, you will start in the ergodic steady-state for MS-model, and in the last regime for break-point models.
 - > 'zero' : Start from zero on all variables (Except for the deterministic exogenous variables)
- 'startingProb' : Either a double or a char:
 > [] : If the model is filtered the starting value is calculated on filtered transition probabilities. Otherwise the

ergodic mean is used.

> double	: A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
> 'ergodic'	: Start from the ergodic transition probabilities.
- 'stabilityTest'	: Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'states'	: Either an integer with the regime to forecast/simulate in, or an object of class nb_ts with the regimes to condition on. The name of the variable must be 'states'.
Caution: For break-point models this is decided by the dates of the breaks, and cannot be set by this option!	
- 'compareToRev'	: Which revision to compare to. Default is final revision, i.e. []. Give 5 to get fifth revision. must be an integer. Keep in mind that this number should be larger than the nSteps input.
- 'compareTo'	: Sometime you want to evaluate the forecast of one variable against another variable which is not part of the model. E.g. if you use the first release of a variable and want to compare it against the final release. To do this you can give a cellstr with size n x 2, where n is the number of variables to change the the actual observations to test against. {'VAR_1','VAR_FINAL',...}.
- 'estimateDensities'	: true or false. true if you want to do a kernel density estimation of the density forecast.
- 'exoProj'	: Tolerate missing exogenous variables by projecting them by a fitted model. Only when the 'condDB' input is empty!
Options are:	
- ''	: No projection are done. Error will be given if some exogenous variables are not given any conditional information. Default.
- 'ar'	: The exogenous variables will be fitted by a univaiate AR model with the lag length chosen with a 'aicc' criterion.
Caution : If density forecast are produced the AR estimates will be bootstrapped, and the exogenous	

variables will be re-sampled based on the bootstrapped values.

Caution : If forecast are being produced recursively with recursively estimated parameters, the forecast/bootstrapping of the exogenous variables will be done based on recursively estimated AR coefficients as well.

- 'exoProjHist' : If this option is set to true, it will first try to get the data on the exogenous variables for the full forecast horizon from the data assigned to the model object. If the data is not found it will extrapolate the exogenous variables by the method given to the 'exoProj' option.

- 'exoProjDiff' : Set to a cell array on the format {'Var1',1, 'Var2',1} to apply 1st difference to the series 'Var1' and 'Var2'. If a variable is not given a value it is not 1st differences (or the value is set to 0).

Caution: Only applies to the nb_exprModel class.

- 'exoProjAR' : Set the number of lags of the AR(X) models when exogenous variables are projected, i.e. set X. Default is nan, i.e. a information criterion is used to select the number of lags (Set this option to nan).

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjDummies' : A cellstr with the dummy variables to include in the AR model when projecting exogenous variables. These dummies are assumed to be predicted by 0.

Caution: Only applies when exoProj is set to 'AR'.

- 'exoProjCalib' : Calibrate the AR process to extrapolate the exogenous variables with. As a 1 x 2*N cell array. N is equal to the number of exogenous variables you extrapolate. An example {'exoVar1',[0,0.8],'exoVar2',[0.2,0.7]}. Here the 'exoVar1' variable is extrapolated with an AR model with a constant term equal to 0 and AR coefficient 0.8, while 'exoVar2' with a constant term equal to 0.2 and AR coefficient 0.8. The exogenous variables not listed will be estimated.

Caution: Only applies when exoProj is set to 'AR'.

- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:

- 'shock' : Name of the shock to match the restriction on the bounds of the given

variable.

- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'inputs' : This is a special input the overwrites all other inputs. This must be given by a struct with fields that include all the other optional options.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution:** You still need to set 'parameterDraws' to the number of draws you want to do.
- 'condAssumption' : Set to 'after' to make the conditional information be interpreted in the states simulated or conditioned on. Otherwise the conditional information is interpreted in the state of end of history.
 - 'seed' : Set simulation seed. Default is 2.0719e+05.

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.
- valid : A logical with size nObj x 1. true at location ii if forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

See also:

[nb_var.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **forecastPerc2Dist** ↑

```
forecastOutput = nb_model_forecast.forecastPerc2Dist(forecastOutput)
```

Description:

Calculates a density from percentiles using kernel estimation.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_var.forecast](#)

Written by Per Bjarne Bye, Kenneth Sæterhagen Paulsen and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► **forecastPerc2ParamDist** ↑

```
[forecastOutput,dist] = nb_model_forecast.forecastPerc2ParamDist(...  
    forecastOutput,distribution,draws,varargin)
```

Description:

Calculates a parameterized density from percentiles using a matching percentiles estimator, and then simulate draws from this distribution. See the nb_distribution.perc2ParamDist function.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.data stores the forecast of the percentiles.
- distribution : The distribution you want to estimate based on the

percentiles. You must have at least as many percentiles as there are parameters!

- draws : Number of draws to make from the estimated distributions.

Optional input:

- 'optimizer' : See doc of the optimizer input to the nb_callOptimizer function. Default is 'fmincon'.
- 'optimset' : See doc of the opt input to the nb_callOptimizer function.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_var.forecast](#), [nb_distribution.perc2ParamDist](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getActual** ↑

```
actual = nb_model_generic.getActual(options,solution,vars,startFcst,...  
nSteps,inputs,nowcast)
```

Description:

Get actual data to compare against the forecast.

Input:

- options : See the (hidden) property estOptions of the nb_model_generic. For real-time models, the last element of this struct array should be used.
- solution : See the property solution of the nb_model_generic object.
- vars : A cellstr with the names of the variables to get the actual data of.
- startFcst : The recursive forecasting periods. Either as a double with the indecies or a cellstr with the dates.
- nSteps : Number of forecasting steps. If empty the sample will not be split.

- inputs : A struct with some options passed to the nb_forecast.getActual function. Optional.
- nowcast : Indicate how many periods there has been produced nowcast of.

Output:

- actual : A nSteps x nVars x nPeriods double with the actual data for each recursive forecast if nSteps is given, otherwise a nobs x nVars double.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getCondDB** ↑

```
[condDB,obj] = getCondDB(obj,endDate,nSteps,exoOnly,recursive,recStart)
```

Description:

Get conditional information on the variables of the model given a end date of estimation. May also provide recursive conditional information.

Caution : If the dataset is of type real-time, this function interprets the endDate input as the estimation end date for the last recursion. It also assumes that each vintage gives rise to one new historical observation on each series of the model! It will also remove the conditional information from the obj.options.data option to make the estimation with real-time data work as supposed to.

Caution : If you do some transformation of your data using the nb_modelData.createVariables you should run this before calling this method!

Input:

- obj : An object of class nb_model_generic.
- endDate : An object of class nb_date or a string representing a date. This option will be interpreted as the end date of estimation or as the end date of the last recursion if recursive estimation is done. See description for a note on real-time estimation.
- nSteps : Number of forecasting steps. As a scalar integer.
- exoOnly : Set to true if you only want the conditional information on the exogenous variables of the model. Default is false.
- recursive : Indicate if you want recursive conditional information or not. If you want to do recursive conditional forecast you should set this input to true, otherwise false. Default

is false.

- `recStart` : An object of class `nb_date` or a string representing a date. This option will be interpreted as the start date of recursive forecast, i.e. the first forecast period of the recursive forecast. Only needed if `recursive` is set to true.

Output:

- `condDB` : As a `nb_ts` or `nb_data` object. Can be given as input to the `condDB` option of the `nb_model_generic.forecast` method.
- `obj` : An object of class `nb_model_generic`. When dealing with real-time data we want to strip the data of forecast before going to estimation, so the `options.data` option may change.

See also:

[nb_modelData.createVariables](#), [nb_var.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_MODEL_GENERIC`

► **getDSGEVARPriorMoments** ↑

```
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR)
[GammaYY, GammaXX, GammaXY] = getDSGEVARPriorMoments(obj, dsgeVAR, varargin)
```

Description:

Get DSGE-VAR prior matrices. See the prior struct returned by `nb_var.priorTemplate('dsge')`.

Input:

- `obj` : An object of class `nb_model_generic`.
- `dsgeVar` : A `nb_var` object set up with options for estimation of a DSGE-VAR.

Optional input:

- `'maxIter'` : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.
- `'tol'` : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

Output:

Let N be number of dependent variables, L the number of lags of the VAR, and C is equal to 1 if a constant is included. otherwise 0.

- GammaYY : A N x N double with the nonstandardized sample moments of the left-hand variables of the VAR.
- GammaXX : A (C + N*L) x (C + N*L) double with the nonstandardized sample moments of the right-hand variables of the VAR.
- GammaXY : A (C + N*L) x N double with the nonstandardized sample moments between the right-hand variables and the left-hand variables of the VAR.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependent** ↑

```
dependent = getDependent(obj)
```

Description:

Get dependent variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- dependent : Either a nb_ts or a nb_cs object with dependent variable(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getDependentNames** ↑

```
dependentNames = getDependentNames(obj)
```

Description:

Get all the unique dependent variables names of a vector of nb_model_generic objects

Input:

- obj : A vector of nb_model_generic objects

Output:

- dependentNames : A cellstr with the unique dependent variables names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getEstimationOptions ↑

```
outOpt = getEstimationOptions(obj)
```

Written by Kenneth Sæterhagen Paulsen

► getEstimationStartDate ↑

```
start = getEstimationStartDate(obj)
```

Description:

Get the start date of the estimation of the model.

Input:

- obj : An object of class nb_model_generic

Output:

- start : An object of class nb_date.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► getFiltered ↑

```
data = getFiltered(obj,type)
data = getFiltered(obj,type,normalize)
data = getFiltered(obj,type,normalize,econometricians,varargin)
```

Description:

Get filtered variables as a nb_ts object. This will include endogenous, exogenous, shocks, states and regime probabilities, if any.

Caution: Model must already be filtered!

Input:

- obj : An object of class nb_model_generic
- type : Either 'filtered', 'smoothed' or 'updated'. Default is 'smoothed'.
- normalize : Normalize the shock to be N(0,1). Default is false.
- econometricians : Get econometricians view of the filtered variables. I.e. the filtered variables of each regime multiplied by the regime probabilities.

Optional input:

- 'reporting' : Either 'stored' or a N x 3 cell array on the format of the reporting property. If 'stored' it will use the reporting property stored in the object. If that is empty, no reporting is done. Default is not to do any reporting.

Output:

- data : A nb_ts object with size nobs x nvars included the filtered variables.

See also:

[nb_dsgen.filter](#), [nb_var.estimate](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getForecast ↑**

```
fcstData = getForecast(obj)
[fcstData,fcstPercData] = getForecast(obj,outputType)
[fcstData,fcstPercData] = getForecast(obj,outputType,includeHist,varargin)
```

Description:

Get the forecast as a nb_ts object.

Caution: Nowcast will be added to the output. Even for those variables with known observations will be returned. For the 'default' and 'graph' output types the nowcast will be abbreviated with period 0,-1 etc. If outputType is set to a date the nowcast will be added as observation date - 1, date - 2, etc.

Input:

- obj : An object of class nb_model_forecast. Must have produced forecast.

- outputType :

> 'default' : The fcstData output is a nb_ts with size nHor x nVars x nPeriods. nPeriods is the number of iterative forecast. The start date of each forecast will be saved in the dataNames property of the nb_ts object.

The fcstPercData output is a struct where the fields are the percentiles/simulations of the density forecast if produced. (Else it is an empty struct). Each field has the same sizes as the fcstData output.

> 'graph' : This option can be used to make it easier to plot the recursive densities.

The fcstData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x 1. Only storing the mean forecast.

The fcstPercData output is a struct where the fields are the recursive forecast periods. Each field has size nHor x nVars x nPerc. Storing the percentiles/simulations of the forecast if they are calculated, or else it is an empty struct.

To graph the recursive forecast use this example:

```
fields = fieldnames(fcstData);
for ii = 1:length(fields)

    fcstMean = fcstData.(fields{ii});
    fcstPerc = fcstPercData.(fields{ii});
    plotter = nb_graph_ts(fcstMean);
    plotter.set('fanDatasets',fcstPerc);
    plotter.graph();

end

> 'date' : A string or a nb_date object with the date of the wanted forecast. E.g. '2012Q1'. The fcstData will be a nb_ts object with size nHor x nVar x nPerc + 1, while fcstPercData will be empty. nPerc will then be the number of percentiles/simulations. Mean will be at last page.
```

For real-time forecast the vintage at the time is returned as the historical data, when includeHist is true.

> 'horizon' : This option will return the forecast as a nPeriods + nHor x nVar x nHor nb_ts object. This option will only return the mean forecast.

If includeHist is true the actual data is added as an additional page of the nb_ts object.

```
Caution : Nowcast will be skipped!
Caution : If the horizon input is set you will set nHor
          to one, and the fcstPercData will be
          non-empty, if density forecast has been
          produced.

- includeHist : Give true (1) to include history in the output. Only an
               options for the 'date' and 'horizon' outputType.
               Default is false (0).
```

Optional inputs:

```
> 'horizon' : The fcstData output will be a nb_ts object with
               size nRec x nVars. While the fcstPercData output
               will be a nb_ts object with size nRec x nVars x nSim.
               You can set the horizon to return by the optional
               input 'horizon'. See the 'timing' option also.
```

Output:

- fcstData : See the input outputType
- fcstPercData : See the input outputType

See also:

[nb_var.plotForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getForecastVariables ↑**

```
vars = getForecastVariables(obj)
```

Description:

Get the variables that the model forecast.

Input:

- obj : An object of class nb_model_forecast.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getHistory** ↑

```
histData = getHistory(obj,vars,date,notSmoothed,type)
```

Description:

Get historical data

Input:

- obj : A scalar nb_model_generic object
- vars : A cellstr with the variables to get. May include shocks/residuals. Only the variables found is returned, i.e. no error is provided if not all variables are found.
- date : For recursively estimated models, the residual and smoothed estimates vary with the date of recursion, so by this option you can get the residual and smoothed estimates of a given recursion.
The same apply for real-time data.
- notSmoothed : Prevent getting history from smoothed estimates.
- type : Either 'smoothed' or 'updated'. Default is 'smoothed'.

Output:

- histData : A nb_ts object with the historical data.

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getIdentification** ↑

```
matrix = getIdentification(obj)
```

Description:

Get restrictions on the structural shock matrix, i.e C in the equation below:

$$y(t) = A*y(t-1) + B*x(t) + C*e(t), \text{ where } e(t) \sim N(0, I).$$

Input:

- obj : A identified scalar nb_var object.

Output:

- matrix : nEndo x nEndo cell array.

See also:

[nb_var.set_identification](#)

Written by Kenneth Sæterhagen Paulsen

► **getIdentifiedResidual** ↑

```
residual = getIdentifiedResidual(obj)
```

Description:

Get identified residual from a estimated nb_var object

If we have a VAR it can be written as (dropping exogenous variables):

$y_t = A*y_{t-1} + e_t$

A identified VAR can be written as:

$y_t = A*y_{t-1} + C*d_t$

Therefore the identified residuals/shocks can be found as:

$d_t = \text{inv}(C)*e_t$

Caution: Will only return the residual that uses the full sample. I.e. when recursive estimation is done, only the residuals from the last estimation will be returned.

Input:

- obj : An object of class nb_var

Output:

- residual : A nb_ts object with the identified residual(s) stored.
As a nobs x nEq x nIdent. Where nIdent is the number of identified C matrices, i.e. when the user have asked to identify more than one matrix (underidentification).

Written by Kenneth Sæterhagen Paulsen

► **getLHSVars** ↑

```
vars = getLHSVars(obj)
vars = getLHSVars(obj, varsIn)
```

Description:

Get all left hand side variables of the model.

For factor models the observable (+ observableFast) are added as well.

The list will be sorted.

Caution : For nb_dsgc object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getModelNames** ↑

names = getModelNames(obj)

Description:

Get model names, if some model names are empty, they will be given the default name 'Model<ii>'.

Input:

- obj : A N x M nb_model_forecast object.

Output:

- names : A N x M cellstr.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_NAME

► **getModelVars** ↑

```
vars = getModelVars(obj)
vars = getModelVars(obj, varsIn)
```

Description:

Get all variables to the model. That include dependent, endogenous and exogenous variables.

For factor models the observable (+ observableFast) are added as well.

Caution : For nb_dsg object only the observables are returned!

Input:

- obj : A scalar nb_model_generic object.
- varsIn : The variables that can be found in the data of the model.
If empty, they are found at obj.options.data.variables.

Caution: Only used in special cases.

Output:

- vars : A cellstr array with the variables of the model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getOriginalVariables** ↑

```
vars = getOriginalVariables(obj)
```

Description:

Get the variables of the original data source assign to the model.

Input:

- obj : An object of class nb_model_generic.

Output:

- vars : A cellstr with the variables of the original data source.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **getPIT** ↑

```

pit                  = getPIT(obj)
[pit,plotter]        = getPIT(obj)
[pit,plotter,pitTest] = getPIT(obj,startDate,endDate,onlyFinal)

```

Description:

Calculate PIT of a density forecast given actual data.

Caution : If recursive density forecast has been produced it will get the PIT from the start date of the recursive forecast and use the density forecast at each periods to produce the pits.

If only one density forecast has been produce it will construct the PITs based on the period from the estimation start date until the end date of forecast horizon.

Caution : Nowcasts will be skipped!

Input:

- obj : A nb_model_forecast object.
- startDate : The start date of the PIT calculations, as a string or a nb_date object. Default is as written above.

Caution : If you do recursive density forecast, this date cannot be before the start date of the recursive forecast (if not onlyFinal is set to true). Otherwise the start date cannot be before the start date of the data stored in the model object.
- endDate : The start date of the PIT calculations, as a string or a nb_date object. Default is to use the last forecast recursion or the date the density is constructed. This input cannot be after that, except when onlyFinal option is set to 1 or only one density is estimated. The end date can then be set to a date as long you have observations on the actual data. (Stored in options.data)
- onlyFinal : Logical. If true the final density forecast is used to calculate the PITs even if recursive density forecast exists. Default is false. If startDate is empty the start date of estimation will be used as default.

Output:

- pit : A nb_data object with size nPeriods x nHor x nVars
- plotter : A nb_graph_data object. please use the graph method, if you want to plot it.
- pitTest : Test for a uniformly distributed PITs using a Pearson chi-squared test. As a 2 x nHor x nVar nb_cs object.

See also:

nb_calculatePIT

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getParameterDrawsMethods** ↑

```
methods = parameterDraws(obj)
```

Description:

Get supported methods for drawing parameters for the given model.

Input:

- obj : A scalar object of a subclass of the nb_model_generic class.
- gui : true or false. Default is false.
- forecast : For forecast. true or false. Default is false.

Output:

- methods : The supported methods for drawing parameters for the given model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getParameters** ↑

```
p = getParameters(obj,varargin)
```

Description:

Same as obj.parameters only that the name and values are concatenated into a nParam x 2 cell matrix instead of a struct.

Input:

- obj : An object of class nb_model_generic.

Optional input

- varargin : If 'struct' is provided among the optional inputs the returned will be a struct where the parameters are the fieldnames and the parameter values as fields. Otherwise a nParam x 2 cell matrix. First column is the parameter names, while the second column is the

```
parameter values ('struct' is not supported if
numel(obj) > 1).

: Give 'headers' to add model names as headers for each
column of the return cell. Only if numel(obj) > 1.

: Give 'double' to return the value as a numerical array.

: Give 'skipBreaks' to not remove any break-point parameters.
```

Output:

- p : See the type input.

See also:

[parameters](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPosteriorDistributions ↑**

```
distr          = getPosteriorDistributions(obj)
[distr,paramNames] = getPosteriorDistributions(obj,'draws',1000)
```

Description:

Get posterior distributions of model.

Input:

- obj : An object of class nb_model_generic.

Optitonal input:

- 'draws' : The number of draws to sample from the posterior to base
the kernel estimation on, if empty all draws are used for
estimation.

Output:

- distr : A 1 x numCoeff nb_distribution object.

- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPredicted ↑**

```
residual = getPredicted(obj)
```

Description:

Get predicted variables from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with predicted variable(s) stored.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getPriorDistributions** ↑

```
[distr,paramNames] = getPriorDistributions(obj)
```

Description:

Get prior distributions of B-VAR model.

Input:

- obj : An object of class nb_var.

Output:

- distr : A 1 x numCoeff nb_distribution object.
- paramNames : A 1 x numCoeff cellstr with the names of the parameters.

Written by Kenneth Å!terhagen Paulsen

► **getRecursiveEstimationGraph** ↑

```
plotter = getRecursiveEstimationGraph(obj)
```

Description:

Get recursive estimation graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphSubPlots method on to get the figure(s).

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph_ts. Use the graphSubPlots method or the nb_graphSubplotGUI class to produce the graphs.

Examples:

```
plotter = getRecursiveEstimationGraph(obj);  
plotter.graphSubPlots();
```

See also:

[nb_graph_ts](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [getRecursiveScore ↑](#)

```
score          = getRecursiveScore(obj,type)  
[score,plotter] = getRecursiveScore(obj,type,startDate,endDate,invert)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_forecast objects.
- type : A string with one of the following:
 - 'RMSE' : Root mean squared error
 - 'MSE' : Mean squared error
 - 'MAE' : Mean absolute error
 - 'MEAN' : Mean error
 - 'STD' : Standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score
- dim : Either 'variables' to get the recursive scores of all the variables forecast by a model (the obj input must be a scalar object), or else the name of the variable to get the score of (must be part of all models!).
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.

- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : true | false. Default true. If true it will invert the score for 'RMSE', 'MSE', 'MAE' and 'STD'.

Output:

- score : A nb_ts object with size nPeriods x nVars x nHorizon if dim equal 'variables', else a nb_ts object with size nPeriods x nObj x nHorizon.
- plotter : A nb_graph_ts object to plot the recursive scores. Use the graph method. One figure per horizon!

See also:

[nb_model_generic.forecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getResidual** ↑

```
residual = getResidual(obj)
```

Description:

Get residual from a estimated nb_model_generic object

Input:

- obj : An object of class nb_model_generic

Output:

- residual : Either a nb_ts or a nb_cs object with residual(s) stored.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualGraph** ↑

```
plotter = getResidualGraph(obj)
```

Description:

Get residual graph from the given estimator. The returned will be an object of class nb_graph, which you can call the graphInfoStruct method on to get the figure(s).

Caution : Will only plot the residual of the last period of recursive estimation.

Caution : For quantile regression this will only report the graph for the median regression.

Input:

- obj : An object of class nb_model_generic (or a subclass of this class).

Output:

- plotter : An object of class nb_graph. Use the graphInfoStruct method or the nb_graphInfoStructGUI class.

Examples:

```
plotter = getResidualGraph(obj);  
plotter.graphInfoStruct();
```

See also:

[nb_graph](#), [nb_graph_ts](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getResidualNames ↑**

```
residualNames = getResidualNames(obj)
```

Description:

Get all the unique residual names of a vector of nb_model_generic object

Input:

- obj : A vector of nb_model_generic objects

Output:

- residualNames : A cellstr with the unique residual names of all the models

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getRoots** ↑

```
roots = getRoots(obj)
```

Description:

Get the eigenvalues of the companion for. If all these eigenvalues are inside the unit circle the model is stable.

If some eigenvalues are on the unit circle the model is non-stationary and you need to use cointegration relations.

If the eigenvalues are outside the unit circle the model is explosive.

Caution : The models must be solved! See the method solve.

Caution : For recursively estimated model each period solution is calculated. See page ModelX (ii), where ii will be the index for the recursive estimation.

Input:

- obj : A vector of nb_model_generic objects

Output:

- roots : A nb_data object. Each page (dataset) gives the roots of each model. This output has three variables 'Real', 'Imaginary' and 'Modulus' storing the given parts of the roots.

- plotter : A nb_graph_data object. Use the graph method.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getScore** ↑

```
scores = getScore(obj,type,allPeriods)
scores = getScore(obj,type,allPeriods,startDate,endDate,invert,... 
                  rollingWindow,lambda)
```

Description:

Get forecasting scores using recursive evaluation.

Input:

- obj : A vector of nb_model_group or nb_model_generic objects.

- type : A string with one of the following:
 - 'RMSE' : One over root mean squared error
 - 'MSE' : One over mean squared error
 - 'MAE' : One over mean absolute error
 - 'MEAN' : One over mean error
 - 'STD' : One over standard error of the forecast error
 - 'ESLS' : Exponential of the sum of the log scores
 - 'EELS' : Exponential of the mean of the log scores
 - 'MLS' : Mean log score

If the object input is a vector, the type input can also be a cellstr array with matching length.

Caution: See comment to the invert input!

- allPeriods : true if you want the scores to be calculated for all periods recursively, or else give false, i.e. only calculate the score for the last period. false is default.
- startDate : The date to begin constructing the score. Can be empty. Either as a string or an object of class nb_date.
- endDate : The date to end constructiong the score. Can be empty. Either as a string or an object of class nb_date.
- invert : false or true. Default is false. Default is to report the score so that high value means good model. E.g. setting type to 'RMSE' and invert to true will give NOT inverted root mean squared error!
- rollingWindow : Set it to a number if the combination weights are to be calculated using a rolling window. Default is empty, i.e. to calculate the weights recursively using the full history at each recursive step.
- lambda : Give the value of the parameter of the exponential decaying weights on past forecast errors when constructing the score. If empty the weights on all past forecast errors are equal.

Output:

- scores : A struct. Each field gives the forecasting evaluation for each model, and for each model the output is given as a nb_data object with size nHor x nVars x nPeriods. nPeriod is 1 if allPeriod is given as false.

See also:

[nb_model_group.combineForecast](#), [nb_model_generic.constructScore](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **getSolution** ↑

```
value = getSolution(obj,type)
```

Description:

Get different objects of the solution.

Input:

- obj : An object of class nb_model_generic
- type : Type of matrix to return. One of:
 - > 'A' : Transition matrix.
 - > 'B' : Impact of exogenous matrix.
 - > 'C' : Impact of shocks matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **getVariablesList** ↑

```
[vars,observables] = getVariablesList(obj,type)
```

Description:

Get list of variables given nb_model_generic object that may analyzed

Input:

- obj : A scalar nb_model_generic object
- type : 'forecast' or 'others'. Default is 'forecast'.

Output:

- vars : A cellstr with the variables the model may forecast.
These are the variables that may be given to the
'varOfInterest' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'.
- observables : A cellstr with the observables the model may forecast.
These are the observables that may be given to the
'observables' input to the forecast method of the
nb_model_generic forecast, when the 'output' input is
set to 'all'. Only some specific class of models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► grangerCausalityTest ↑

```
[test,pval,results] = grangerCausalityTest(obj,precision)
```

Description:

Test for granger causality between variables using a joint F-test that all the parameters of the lags of a variable y is zero in the equation where x is the dependent variable. If the hypothesis can be rejected we say that y granger cause x.

Caution: If recursive estimation is done it will anyway only do the test using the parameters estimated over the full sample.

Input:

- obj : A 1 x 1 nb_var object
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : The test statistics. As a nDep x nDep double. The diagonal will return nan.
- pval : The p-value of the test statistics. As a nDep x nDep double. The diagonal will return nan.
- results : A string with the printed results of the test.

Written by Kenneth Sæterhagen Paulsen

► graphCorrelation ↑

```
plotter = nb_model_generic.graphCorrelation(variable,movingVar,...  
c,cEmp,ac1,ac1Emp,ac2,ac2Emp,...)
```

Description:

Graph model correlations against empirical.

Input:

- variable : The main variable, as a string.
- movingVar : A cellstr or char with the variables to plot the covariance/correlations with the main variable.

Optional input:

- 'fan' : true or false. Give true to make shaded areas between the
- The first input must be the c output from nb_model_generic.simulatedMoments (SIM) or nb_model_generic.theoreticalMoments (THEO), the second input must be the c output from nb_model_generic.empiricalMoments (EMP), the third input can be the acl output from SIM or THEO, the fourth input can be the acl output from EMP, and so on.

Caution : The inputs must be given as nb_cs objects.

Caution : Inputs must come in pairs.

Output:

- plotter : An object of class nb_graph_cs. Use the graph method or the nb_graphPagesGUI class.

See also:

[nb_var.simulatedMoments](#), [nb_var.theoreticalMoments](#)
[nb_var.empiricalMoments](#), [nb_graphPagesGUI](#), [nb_graph_cs](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **handleCondInfo** ↑

ret = handleCondInfo(obj)

Description:

Check if a nb_model_estimate object handle conditional info.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle conditional info.

Written by Kenneth Sæterhagen Paulsen

► **handleMissing** ↑

```
ret = handleMissing(obj)
```

Description:

Check if a nb_model_estimate object handle missing observations.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model handle missing data.

Written by Kenneth Sæterhagen Paulsen

► **help ↑**

```
helpText = nb_var.help
helpText = nb_var.help(option)
helpText = nb_var.help(option,maxChars)
```

Description:

Get the help on the different model options.

Input:

- option : Either 'all' or the name of the option you want to get the help on.
- maxChars : The max number of chars in the printout of (approx). This applies to the second column of the printout. Default is 40.

Output:

- helpText : A char with the help.

Written by Kenneth Sæterhagen Paulsen

► **identDraws2Solutions ↑**

```
sol = identDraws2Solutions(obj)
```

Description:

This function assumes that you have used the set_identification function with the input 'type' set to 'combination'.

Input:

- obj : An object of class nb_var.

Output:

- sol : A struct storing the solution of the model. Same format as the solution property of the nb_var class.

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
obj = initialize(template)
```

Description:

Initialize multiple model objects, see output from the template method of each subclass of the nb_model_generic class.

Input:

- objClass : The name of any of the subclasses of the nb_model_generic class. As a string.
- template : The output from the template method of each subclass of the nb_model_generic class. Multiple model objects will be initialized if the num input that method is greater than one.

Output:

- obj : An object which is of nb_model_generic

Examples:

```
template = nb_var.template(4);
obj      = nb_model_generic.initialize('nb_var',template);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **interpretForecast** ↑

```
[obj,lik,normDiff] = interpretForecast(obj,varargin)
```

Description:

Input:

- obj : A vector of nb_model_generic objects.

Optional input:

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if 'parallel' is set to true.
- 'fcstDB' : A nSteps x nVars nb_ts object. These are the forecast to be interpreted by the model represented by obj.
- 'model' : A nb_model_generic object that produces some forecast to be interpreted by the model represented by obj. The forecast method is used to produce these forecast.
- 'nSteps' : Number of forecasting steps. As a 1x1 double. Default is 8 if the 'model' options is given, otherwise this option does not apply.
- 'optimizer' : The optimizer used to solve the problem. Default is 'fmincon'. Use the nb_getOptimizers method to get optimizer to select from.
- 'optimset' : Set options for forecast matching. See the nb_getDefaultOptimset function. Which fields that are important depend on the 'optimizer' input.
- 'parallel' : Give true if you want to do the forecast in parallel. I.e. spread models to different threads. Default is false. Default is 1.
- 'parameters' : To use parameters to interpret the forecast, you need to list it using this input. As a 1 x nParam cellstr. Caution: You must assign some prior distributions using the setPrior method to be able to utilize this input. It is also only supported if obj is of class nb_dsgc.
- 'periods' : The number of periods that you let innovations from the model represented by obj be active.
- 'startDate' : The start date of forecast. Must be a string or an object of class nb_date. Is empty the estim_end_date + 1 will be used. This option only applies if the 'model' option is provided.
- 'varOfInterest' : A char with the variable to produce the forecast of. Default is '', i.e. all variables. Can also be given a cellstr with the variables to forecast. Variables provided which is not part of the model will be discarded and no error will be given!
- 'varsToMatch' : If the 'model' input is given these are the

variables that are matched, otherwise these are taken from the 'fcstDB' option.

- 'scale' : Sets the relative weight between the concern to match the forecast from the alternative model, and the likelihood of the innovation that are used to match those restrictions. $\text{norm}(\text{fcst} - \text{fcstA}) + \text{scale} * \text{logFunc}$, where logFunc is either the minus log likelihood or the minus the log posterior. A number greater than or equal to 0. Default is 0.
- 'weights' : A nb_ts object with same timespan and variables as 'fcstDB' or the forecast output from the 'model' input. Each row should sum to 1, and will be used to weight the concern of matching the different forecast. Default is equal weights.

See also:

[nb_var.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **irf ↑**

[irfs, irfsBands, plotter] = irf(obj, varargin)

Description:

Create impulse responses of the given vector of nb_model_generic objects

Caution : If recursive estimation is done, only the last estimated model will be used.

Input:

- obj : A vector of nb_model_generic objects

Optional input:

- 'addSS' : Set to false to not add the steady-state to IRF of nb_dsgen objects.
- 'compare' : Give true if you have a scalar nb_dsgen model and you want to graph the irfs from the different regimes from a NBToolbox solved model against eachother. Default is false.
- 'compareShocks' : Give true if you want to plot all the IRFs of different shocks in the same graphs (one separate graph for each variable). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'compareVars' : Give true if you want to plot all the IRFs of different variables in the same graphs (one separate graph for each shock). In this case you need a scalar nb_model_generic object as the obj input!

Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs in this case.

- 'continue' : true or false. If true it will continue from a paused call to this method, or else it will start from the beginning.

Caution : All the inputs will be taken from the paused run!

- 'cores' : The number of cores to open, as an integer. Default is to open max number of cores available. Only an option if either 'parallel' or 'parallelL' is set to true.

- 'draws' : Number of draws for calculating girf. Default is 1000. For Markov switching models this will set the number of simulated paths of states. For more see the 'type' input.

- 'factor' : A cell array with the factors to multiply the irf of the individual model variables.
I.e. {'var1',100,...} or {{'var1','var2'},100,...}

If the string starts with a asterisk you will multiply all the variables that contain that string with the given factor.
E.g. {'*_GAP',100}

- 'fanPerc' : This options sets the error band percentiles of the graph, when the 'perc' input is empty. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

- 'irfCompare' : A struct on the same format as the irfs output from this function with the IRFs to compare to.

- 'levelMethod' : One of the following:

```

> 'cumulative product' : cumprod(1 + X,1)
> 'cumulative sum' : cumsum(X,1)
> 'cumulative product (log)' : log(cumprod(1 + X,1))
> 'cumulative sum (exponential)' : exp(cumsum(X,1))
> 'cumulative product (%)' : cumprod(1 + X/100,1)
> 'cumulative sum (%)' : cumsum(X/100,1)
> 'cumulative product (log) (%)' : log(cumprod(1 + X/100,1))
> 'cumulative sum (exponential) (%)' : exp(cumsum(X/100,1))
> '4 period growth (log approx)' : nb_msum(X,3)

- 'method' : The selected method to create error bands.
  Default is ''. See help on the method input of the
  nb_model_generic.parameterDraws method for more
  this input. An extra option is:

    > 'identDraws' : Uses the draws of the matrix with
      the map from structural shocks to dependent
      variables. See nb_var.set_identification. Of
      course this option only makes sense for VARs
      identified with sign restrictions.

- 'normalize' : A 1 x 3 cell. First element must be the variable
  name as a string. The second element must be the
  value to normalize to, as an integer. While the
  third element is the period to be normalized, if
  set to inf, it will normalize the max impact
  period.

  E.g. {'Var',1,2}, {'Var',1,inf}

  Caution: 2 means observation 2 of the IRF, and as
  the IRF start at period 0, this means that
  observation 2 is period 1.

- 'normalizeTo' : 'draws' or 'mean'. 'draws' normalize each draw of
  irfs, while 'mean' normalize the mean and scale
  percentiles/other draws accordingly. Default is
  'draws'.

  Caution: Setting this to 'mean' will not work for
  reported variables that is not measured
  as % deviation from steady-state/mean.

- 'newReplic' : When out of parameter draws, this factor decides how
  many new draws are going to be made. Default is 0.1.
  I.e. 1/10 of the replic input.

- 'parallel' : Give true if you want to do the irfs in parallel.
  This option will parallelize over models. Default
  is false.

```

- 'parallelL' : Give true if you want to do the irfs in parallel. This option will parallelize over parameter simulations. Only an option if numel(obj) == 1. Default is false.
- 'pause' : true or false, Enable or disable pause option when calculating the irfs. Default is true.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. The input must be given as the coverage, and not the percentiles itself. E.g. [0.3,0.5,0.7,0.9]. 0.3 means to return the two percentiles 35 and 65, which will cover 30% of the distribution around the median. So [0.3,0.5,0.7,0.9] will get the percentiles [5,15,25,35,65,75,85,95], i.e. this will restrict the percentiles to be symmetric. If empty (default) all the simulation will be returned. (The graph will give percentiles specified by the 'fanPerc' input (0.68).)
- 'periods' : Number of periods of the impulse responses.
- 'plotSS' : Set to true to plot the steady-state in the IRF graphs. Default is false. Only for nb_dsg objects.
- 'plotDevInitSS' : Set to true to plot the IRFs as deviation from the initial steady state. Only an option if 'plotSS' is set to true. Only for nb_dsg objects.
- 'replic' : The number of parameter draws.

Caution: 'perc' must be used to produce error bands!

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'settings' : Extra graphs settings given to nb_plots function. Must be a cell.
- 'shocks' : Which shocks to create impulse responses of. Default is the residuals defined by the first model.

For Markov switching or break point models it may be wanted to only run a IRF when switching the state. To do so set this option to {'states'}. You should also set 'startingValues' to 'steadystate(r)', where r is the regime you start in. For a Markov switching model also set 'startingProb' to the same r. This is only an option if 'type' is set to 'irf'.
- 'sign' : Sign of the impulse. Either 1 or -1. Can also be a vector of the same size as 'shocks' input.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default

```

is false.

- 'startingProb' : Either a double or a char (Only for Markov-switching
models):

    > scalar      : Select the the regime to take the
                     initial transition probabilities
                     from.

    > double      : A draws x nRegime or a 1 x
                     nRegime double. draws refer to
                     the number set by the 'draws'
                     input.

    > 'ergodic'   : Start from the ergodic transition
                     probabilities. Default for 'irf'.

    > 'random'    : Randomize the starting values
                     using the simulated starting
                     points. Default for 'girf'.

- 'startingValues' : Either a double or a char:

    > double      : A nVar x 1 double.

    > 'steadystate' : Start from the steady state. If you are dealing
                      with a MS-model or a break-point model you can
                      indicate the regime by 'steadystate(regime)'. E.g.
                      to start in regime 1 give; 'steadystate(1)'. For
                      MS - models the default is the ergodic mean
                      (default as long as 'girf' is not selected as
                      'type'), while for break-point models it is to
                      start from the first regime.

    > 'zero'       : Start from zero on all variables.

    > 'random'    : Randomize the starting values using the simulated
                     starting points. Default when 'type' set to
                     'girf'.

- 'states'       : Either an integer with the states to produce irf in,
                     or a double with the states to condition on. Must
                     have size periods x 1 in the last case. Applies to
                     both MS - models and break-point models.

- 'type'         : 'girf' or 'irf'. (If empty 'irf' will be used)

    > 'girf' : Generlized impulse response function
               calculated as the mean difference between
               shocking the model with a one period shock
               (1 std) and no shock at all. Use the 'draws'
               input to set the number of simulations to use
               to base the calculation on. This type of
               irfs must be used for non-linear models.

    > 'irf'  : Standard irf for linear models. Calculated by
               shocking the model with a one period shock
               (1 std).

```

- 'variables' : The variables to create impulse responses of.
Default is the dependent variables defined by the first model.

Caution: The variables reported by the reporting property may also be asked for here!

- 'variablesLevel' : As a cellstr. The variables to transform to level using the method specified by 'levelMethod'.

Output:

- irfs : The (central) impulse response function stored in a structure of nb_ts object. Each field stores the the impulse responses of each shock. Where the variables of the nb_ts object is the impulse responses of the wanted endogenous variables.

I.e. the impuls responses to the shock 'E_X_NW' is stored in irfs.'('E_X_NW'). Which will then be a nb_ts object of all the wanted variables.

Caution: If more nb_model_generic objects are given, the responses from each model are saved as different datasets (pages) of the nb_ts object.

Caution: Each nb_model_generic object can have different variables and shocks.

- irfsBands : A struct with the shocks as fieldnames. Each field store a nb_ts object with the error bands of all variables. The pages of the nb_ts object is the percentiles (from lower to upper)/simulations.

Caution: Only the first model will get error bands!

- plotter : An object of class nb_graph_ts.
 - > 'compareShocks' set to false (default)
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
 - > 'compareShocks' set to true
Use the graphSubPlots method or the nb_graphSubPlotGUI class to produce the graphs.
- obj : If the process has been paused, some temporary output will be stored in the object, and if you want to continue at a later stage you can take up from this point using this returned output. If not paused this output is empty.

See also:

[nb_var.parameterDraws](#), [nb_irfEngine](#), [nb_irfEngine.irfPoint](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isDensityForecast** ↑

```
ret = isDensityForecast(obj)
```

Description:

Has the nb_model_forecast object produced density forecast or not?

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **isFiltered** ↑

```
ret = isFiltered(obj)
```

Description:

Test if a nb_dsgc object is filtered or not.

Input:

- obj : An object of class nb_dsgc.

- tested : Either 'smoothed', 'filtered' or 'updated'. Default is 'smoothed'.

Output:

- ret : 1 if filtered, otherwise 0.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isIdentified** ↑

```
ret = isIdentified(obj)
```

Description:

Is the nb_var object identified or not.

Input:

- obj : A nb_var object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► **isMS** ↑

ret = isMS(obj)

Description:

Is the nb_model_genric object a Markov switching model or not.

Caution : Model must be solved, otherwise ret is always false!

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isMixedFrequency** ↑

ret = isMixedFrequency(obj)

Description:

Does the model support mixed frequency?

Input:

- obj : An object of class nb_model_generic.

Output:

- ret : True or false.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isNB ↑

```
ret = isNB(obj)
```

Description:

This will return true for all objects except nb_dsge objects!

For nb_dsge objects it will return true if the DSGE model is parsed/solved using the NB Toolbox solver for DSGE models, otherwise it will return false.

Input:

- obj : A M x N x Q nb_model_generic object.

Output:

- ret : A M x N x Q logical. An element is set to true if the model uses a NB toolbox solver.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► isStateSpaceModel ↑

```
ret = isStateSpaceModel(obj)
```

Description:

Check if a nb_model_generic object is a state-space model or not.

Input:

- obj : A scalar nb_model_generic object.

Output:

- ret : true or false. true if the estimation settings is so that the solution to the model is a state-space model.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isStatic** ↑

```
ret = isStatic(obj)
```

Description:

Check if a nb_model_estimate object is a static model or not.

Input:

- obj : A scalar nb_model_etimate object.

Output:

- ret : true or false. true if the estimation settings is so that the model is static.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **isbayesian** ↑

```
ret = isbayesian(obj)
```

Description:

Can the nb_model_genric object be estimated using bayesian methods (using the NB Toolbox) or not.

Input:

- obj : A nb_model_genric object (matrix).

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isestimated** ↑

```
ret = isestimated(obj)
```

Description:

Is the nb_model_genric object empty (not estimated) or not.

Caution: For DSGE model it will return true also if the model is only filtered.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **isforecasted ↑**

ret = isforecasted(obj)

Description:

Is the nb_model_forecast object forecasted or not.

Input:

- obj : A nb_model_forecast object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **issolved ↑**

ret = issolved(obj)

Description:

Is the nb_model_genric object solved or not.

Input:

- obj : A nb_model_genric object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► jointPredictionBands ↑

```
[bands,plotter] = jointPredictionBands(obj,varargin)
```

Description:

Calculate joint predication bands.

Input:

obj : An scalar nb_model_generic object

Optional inputs:

- 'vars' : A cellstr with the variables to return the calculated joint prediction bands of.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9] (default) or 0.9. Cannot be empty!
- 'date' : If recursive forecast has been produced, you can use this option to choose which of the recursive forecast to construct the joint prediction bands of. Default is empty, i.e. use the last forecast.
- 'method' : Choose between:
 - > 'kolsrud' : See Akram et al (2016), Joint prediction bands for macroeconomic risk management
 - > 'copula' : Using a copula likelihood approach.
 - > 'mostlik' : Group the paths based on how likely they are.
- 'nSteps' : Number of forecasting steps to use when constructing the error bands. If empty (default) all forecasting steps stored in the object is used.

Output:

- JPB : An nb_ts object storing the joint prediction bands. The percentiles are stored as datasets. See the

dataNames property for which page represent which percentile.

The data of the nb_ts object has size nSteps x nVars x nPerc.

- plotter : A nb_graph_ts object. Use the graphSubPlots method to produce the graphs.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **loadPosterior** ↑

```
posterior = loadPosterior(obj)
```

Description:

Load posterior of the estimated model. Bayesian models only!

Input:

- obj : An object of class nb_model_generic.

Output:

- posterior : A struct with the posterior output (The format will depend on the sampling method used).

See also:

[nb_model_estimate.estimate](#), [nb_var.assignPosteriorDraws](#)
[nb_model_sampling.sample](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **mcfRestriction** ↑

```
f = mcfRestriction(obj,type,restriction)
```

Description:

Create restriction function for use in the monteCarloFiltering method

Caution: To combine more restriction you can use `@(x)f1(x)&&f2(x)`, where f1 and f2 are the output from this function.

Input:

- obj : An object of class nb_model_generic.
- type : The type of restriction. Either 'irf', 'corr' or 'cov'.
- restriction : Depends on the type input:
 - > 'irf' : A N x 4 cell, where the elements of each row are:
 1. Name of the shock. 'E_X'
 2. Name of the variable. 'X'
 3. Horizon. E.g. 1 or 1:3.
 4. Restriction. E.g. @(x)gt(x,0), @(x)gt(x,0)||lt(x,1), i.e. a function_handle that takes a scalar double as input and a scalar logical as output.
 - > 'rirf' : A N x 7 cell, where the elements of each row are:
 1. Name of the shock 1. 'E_X'
 2. Name of the variable 1. 'X'
 3. Horizon of variable 1. E.g. 1.
 4. Name of the shock 2. 'E_Y'
 5. Name of the variable 2. 'Y'
 6. Horizon of variable 2. E.g. 2.
 7. Restriction. E.g. @(x,y)gt(x,y), @(x,y)gt(x-y,0)||lt(x-y,1), i.e. a function_handle that takes a two scalar double as inputs and a scalar logical as output.
 - > 'corr' : A N x 4 cell, where the elements of each row are:
 1. Name of the first variable.
 2. Name of the second variable.
 3. Number of periods to lag the 2. variable.
 4. Same as 4 for 'irf'.

E.g. {'Var1','Var1',1,@(x)gt(x,0.1)}, to test a restriction on the autocorrelation at lag for variable 'Var1'.

E.g. {'Var1','Var2',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous correlation between 'Var1' and 'Var2'.
 - > 'cov' : Same as for 'corr'.

E.g. {'Var1','Var1',0,@(x)lt(x,0.1)}, to test a restriction on the contemporaneous variance.
 - > 'ss' : A N x 2 cell, where the elements of each row are:
 1. The expression using any of the endogenous variables of the model.
 2. Same as 4 for 'irf'.

Output:

- f : A function_handle that can be used as an input to the monteCarloFiltering method.

See also:

[nb_var.monteCarloFiltering](#), [nb_var.irf](#)
[nb_var.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [mincerZarnowitzTest](#) ↑

[test,pval,res] = mincerZarnowitzTest(obj,precision)

Description:

Do the Mincer-Zarnowitz test for bias in the recursive forecast.

For real-time forecast the final revision of the actual data is used to test against.

Caution: Testing of nowcasts is not supported, and are skipped!

Input:

- obj : An object of class nb_model_generic. You need to call one of the forecasting functions first!
- precision : The precision of the printed result. As a string. Default is '%8.6f'.

Output:

- test : A nb_ts object with the test statistic. As a nHor x nModel x nVar nb_ts object.
- pval : A nb_ts object with the p-values of the test. As a nHor x nModel x nVar nb_ts object.
- res : A char with the printout of the test.

See also:

[nb_var.uncondForecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► monteCarloFiltering ↑

```
paramD = monteCarloFiltering(obj,varargin)
[paramD,values,solvingFailed,objAcc] = monteCarloFiltering(obj,varargin)
```

Description:

Draws parameter values from a compact space, which is spesified with some lower and upper bounds. For each draws the model is tried to be solved, if success the behavior of the model is tested and if that is a success that model is returned.

The draws are made using the function nb_monteCarloSim.

Input:

- obj : A scalar nb_model_generic object.

Optional input:

- 'draws' : The number of draws from the compact set to be tested. Default is 10000.
- 'func' : A function_handle or a string with the name of a function. The function should take one input, and that input is assumed to be an object of class nb_model_generic. The (only) output must be either true or false, if 'output' is set to 'logical' (default), or it must be a scalar double, if 'output' is set to 'double'. Return true if the model fit a certain criteria.
- 'method' : The method used to draw the monte carlo simulated parameters. See the method input to the nb_monteCarloSim function for the supported values. Default is 'latin'.
- 'parallel' : Give true to run the MCF in parallel. Default is false.
- 'parameters' : A 1 x N cellstr with the parameter names to be drawn from. Must be part of model, and must be provided!
- 'lowerBound' : A 1 x N double with the lower bound on the parameters of interest.
- 'upperBound' : A 1 x N double with the upper bound on the parameters of interest.
- 'waitbar' : true or false. Default is true.
- 'output' : Either 'double' or 'logical'. Default is 'logical'.
- 'seed' : Set the seed to use to draw the monte carlo simulated parameter sets. Default is 1. Seed is set back to old state after this method is called!

Output:

- paramD : The draws made from the parameter space. As a draws x N double. Use paramD(success,:) to get the accepted draws.
- values : A N x 1 logical/double. An element is true/not nan if the model where solved and the test function returned a value.
- solvingFailed : A N x 1 logical. An element is true if the model could not be solved.
- objAcc : A 1 x nAcc vector of nb_model_generic objects representing the accepted models.

See also:

function_handle, nb_var.mcfRestriction, nb_var.solve
nb_var.assignParameters

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **parameterDraws** ↑

```
out = parameterDraws(obj)
out = parameterDraws(obj,draws,method,output,stable)
```

Description:

Make either posterior draws or bootstrapped draws from the distribution of the estimated parameters.

Caution : Not yet supported for recursively estimated models.

Input:

- obj : An object of a subclass of the nb_model_generic class.
- draws : Number of draws, as an integer. Default is 1000.
- method : The selected method to make draws.
 - > 'asymptotic' : Draw from the confidence set using the assumption of asymptotic normality. See the nb_mlEstimator.drawParameters for more on this option. Applies to models estimated with maximum likelihood.
 - > 'bootstrap' : Create artificial data to bootstrap the estimated parameters. Default for models estimated with classical methods.

```

> 'wildBootstrap'      : Create artificial data by wild
                           bootstrap, and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'blockBootstrap'    : Create artificial data by
                           non-overlapping block bootstrap,
                           and then "draw" paramters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'mblockBootstrap'   : Create artificial data by
                           overlapping block bootstrap,
                           and then "draw" paramters based
                           on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'rblockBootstrap'   : Create artificial data by
                           overlapping random block length
                           bootstrap, and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'wildBlockBootstrap' : Create artificial data by wild
                           overlapping random block length
                           bootstrap., and then "draw" paramters
                           based on estimation on these data.
                           Only an option for models estimated
                           with classical methods.

> 'copulaBootstrap'    : Uses a copula approach to draw residual
                           that are autocorrelated, Does not handle
                           heteroscedasticity. Only an option for
                           models estimated with classical methods.

> 'posterior'          : Posterior draws. Only for models
                           estimated with bayesian methods.
                           Default for models estimated with
                           bayesian methods.

- output : 'param' | 'solution' | 'object'. See output section for more
  on this option.

- stable : Give true if you want to discard the parameter draws
  that give rise to an unstable model. Default is false.

```

Optional input:

- 'parallel' : Run in parallel. true or false.
- 'cores' : Number of cores to use. Default is to use all cores
 available.
- 'newDraws' : When out of parameter draws, this factor decides how

many new draws are going to be made. Default is 0.1.
I.e. 1/10 of the draws input.

Caution: When setting 'parallel' to true on MATLAB version later than R2017B this number will be the number of new draws, and not the factor!!! E.g. set it to 100. If it is given as a number less than 1, it will by default be set to 100.

- 'initialDraws' : When drawing parameters this factor decides how many many draws that are produced before solving the model. Default is 1. I.e. it is equal to draws input. For VAR models with underidentification it may be a good idea to set this to a value > 1 as you may drop a lot of parameters when identification fails. Default is 1.

Output:

- out : The output depend on the input output:
> 'param' : Default. A struct with the following fields:
 beta : A nPar x nEq x draws double with the estimated parameters.
 sigma : A nEq x nEq x draws double with the estimated covariance matrix.

For factor models these fields are also returned:
 lambda : Estimated coefficients of the observation equation. As a nFac x nObservables x draws double.
 R : Estimated covariance matrix of the observation equation. As a nFac x nFac x draws double.
 factors : Draws of the factors as a T x nFac x draws double.

> 'solution' : Get the solution of the model for each draw from the distribution of parameters. The output will be a struct with size 1 x draws. The struct will have the same format as the solution property of the underlying object. I.e. only one output!

> 'object' : Get the draws as a 1 x draws nb_model_generic object. Each element will be a model representing a given draw from the distribution of the parameters. I.e. only one output!

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► parameterIntervals ↑

```
ci = parameterIntervals(obj,alpha)
```

Description:

Get confidence intervals (classic) or probability intervals (bayesian).

Confidence intervals are constructed as:

```
beta + (+/-nb_distribution.t_icdf(alpha/2))*stdBeta.
```

Probability intervals are constructed as the per

Input:

- obj : A scalar object of class nb_model_generic.
- alpha : Confidence level/probability band.

Output:

- ci : A nPar x 3 cell matrix.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► plotForecast ↑

```
plotter = plotForecast(obj)
[plotter,plotFunction2Use] = plotForecast(obj,type,startDate, ...
increment,varargin)
```

Description:

Plot forecast.

Caution density forecast will only be displayed for the first model in the vector of nb_model_forecast objects

Input:

- obj : A vector of nb_model_forecast objects
- type : Type of plot, as a string;
 - > 'default' : Plot the forecast for the last period of estimation and ahead. Possibly with density for the first model in the model vector. Can have different variables. If startDate is given this date will be the start of the graph instead.

```

> 'hairyplot' : Plot the recursive point forecast in the same
graph as the actual data.

- startDate : A string with the start date of the forecast to plot. As a
string or an object of class nb_date. Only an option when
type is 'default'. Default is to plot the latest forecast
produced.

- increment : Number of periods between the hairs when doing a
hairy-plot. Default is 1.

```

Optional input:

```

- 'type'      : If the history is of some variables are filtered, you can
use this input to choose to plot the smoothed or updated
estimates of the filtered variables. Either 'smoothed' or
'updated'. Default is 'smoothed'.

```

Output:

```

- plotter       : An object of class nb_graph. Use the graphSubPlots
method or the nb_graphSubPlotGUI class.

- plotFunction2Use : A string with the name of the method to call on the
plotter object to produce the graphs.

```

See also:

[nb_model_generic.uncondForecast](#), [nb_model_generic.forecast](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotForecastDensity** ↑

```
plotter = plotForecastDensity(obj,date,variable)
```

Description:

Plot the density forecast at a given date for a given variable. It will
be compared to the normal distribution with the same mean and std.

Caution : If the model produce nowcast this nowcast will start before the
provided date. See the dataNames property of the DB
property of the returned plotter object. If the model has
produced nowcast the names will 'horizon0' (a variable lag one
period), 'horizon-1' (a variable lag two periods), etc.

Input:

- obj : A 1x1 nb_model_forecast object.
- date : A string or nb_date object with the date of the wanted forecast.
- variable : A string with the variable of interest.

Output:

plotter = A nb_graph_data object. Use the graph method to produce the figure, or the nb_graphPagesGUI class.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_FORECAST

► **plotMCF ↑**

plotter = nb_model_generic.plotMCF(paramD,params,lowerBound,...
upperBound,method)

Description:

Plot parameter draws from Monte Carlo filtering.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- lowerBound : A 1 x N double with the lower bound on the parameters of interest.
- upperBound : A 1 x N double with the upper bound on the parameters of interest.
- method : Either 'biplot' or 'allInOne'. 'biPlot' will plot all pairwise combination against each other in a scatter plot. 'allInOne' will plot the parameters on the x-axis and the accepted parameter values of each of the parameters on the y-axis. Default is 'allInOne'.

Output:

- plotter : > 'allInOne' : A nb_graph_cs object. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- > 'biplot' : A vector of nb_graph_data objects with size equal to the number of pairwise combination of the parameters that can be made. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_var.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFDistTest** ↑

```
plotter = nb_model_generic.plotMCFDistTest(paramD,params,values,test)
```

Description:

Split parameter sets from monte carlo filtering into two pools of sets, one for the parameter sets that lead to success, and the other for those that did not. Then for each pool the CDF of each parameter is compared in graph. In the title the p-value of the selected test is provided.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- test : Name of the test to apply. Either 'smirnov' or 'cucconi' (default). Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_var.monteCarloFiltering](#), [nb_cucconiTest](#), [nb_smirnovTest](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotMCFValues** ↑

```
plotter = nb_model_generic.plotMCFValues(paramD,params,nameValue)
```

Description:

Plot parameter draws from Monte Carlo filtering and matching results.

Input:

- paramD : A draws x N double with the parameter draws. E.g. use paramD(success,:) from the output of the monteCarloFiltering method. N is the number of parameters.
- params : A 1 x N cellstr with the names of the parameters.
- values : A draws x 1 double with the values of the monte carlo filtering.
- nameValue : Name of the values used in the graphs produced. Default is 'Value'. Must be a one line char.

Output:

- plotter : A vector of nb_graph_data objects with size 1 x N. Use the graph method or the nb_graphMultiGUI class to produce the graphs.

See also:

[nb_var.monteCarloFiltering](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [plotPosterioriors](#) ↑

```
plotter = plotPosterioriors(obj)
plotter = plotPosterioriors(obj,varargin)
```

Description:

Plot posteriors (and priors) drawn during estimation. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model that is estimated.

Optitonal input:

- 'prior' : Give this string as an input to compare posterior draws with the prior distributions (if available).
- 'updated' : Give this string as an input to compare posterior draws with the updated prior distributions (if available).

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.
- 'draws' : The number of draws to sample from the posterior to base the kernel estimation on, if empty all draws are used for estimation. If 'updated' is given as input, this will also set the the same options for the updated prior!

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the posteriors on screen.
Caution: If 'subplot' is given it will return a scalar nb_graph_data object. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the posteriors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)
[nb_var.getPosteriorDistributions](#)

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **plotPriors ↑**

plotter = plotPriors(obj,varargin)

Description:

Plot priors. Only for bayesian models.

Input:

- obj : A scalar object of class nb_model_generic. It must represent a bayesian model with a set prior.

Optitonal input:

- 'subplot' : Give this string as an input to plot the priors and posteriors in subplots instead of one plot per parameter.

Output:

- plotter : A 1 x numCoeff vector of objects of class nb_graph_data. Use either the graph method or the nb_graphMultiGUI class to plot the priors on screen.

Caution: If 'subplot' is provided a scalar nb_graph_data object is returned. Use the graphSubPlots method or the nb_graphSubPlotGUI class to plot the priors on screen.

See also:

[nb_graph_data](#), [nb_graphMultiGUI](#), [nb_graphSubPlotGUI](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print** ↑

```
printed = print(obj)
```

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print_estimation_results](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **printCov** ↑

```
printed = printCov(obj)
```

Description:

Get the estimated covariance matrix results printed to a char array.

Caution: Some estimators may not support this option.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_model_estimate.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **print_estimation_results** ↑

printed = print_estimation_results(obj)

Description:

Get the estimation result printed to a char array.

Input:

- obj : A vector of nb_model_estimate objects.

Output:

- printed : A char array with the estimation results.

See also:

[nb_var.print](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **priorTemplate** ↑

prior = nb_var.priorTemplate()
prior = nb_var.priorTemplate(type)
prior = nb_var.priorTemplate(type, num)

Description:

Construct a struct which can be given to the method setPrior.

The structure provided the user the possibility to set different prior options.

Caution: Hyperpriors will only be used if both the empirical and hyperprior options are set to true!

Input:

- type : A string;

- 'glp' : This is the prior used in the paper by Giannone, Lenza and Primiceri (2014). Cannot apply block exogenous variables with this prior.

> 'ARcoeff' : Hyperparameter on first lag coefficient of each equation. Default is 1.

> 'coeff' : See same options for the 'minnesota' prior.

> 'lambda' : Hyperparameter that controls the overall tightness of this prior. Default is 0.2.

> 'Vc' : Hyperparamter on exogenous variables. Default is 1e7.

> 'S_scale' : Prior scale of prior on sigma. Default is 1, i.e. OLS.

- 'glpMF' : Same as 'glp', but as we now need a gibbs sampler the below options are added. This prior option supports missing observations. Cannot apply block exogenous variables with this prior. The 'ARcoeff' defaults to 0.9 in this case, as the model must be stationary at the prior to run the kalman filter.

> 'burn' : How many draws that should be used as burn in. Default is 500.

> 'thin' : Every k draws are kept when doing gibbs. This options sets k. Default is 2. Increase this option to prevent autocorrelated draws. See nb_model_generic.checkPosteriors to test for autocorrelated draws.

- 'jeffrey' : Diffuse jeffrey prior. No options.

- 'minnesota' : Minnesota prior options. See page 6 of Koop and Korobilis (2009). Output fields:

> 'a_bar_1' : Hyperparamter on own lags. Default is 0.5.

> 'a_bar_2' : Hyperparamter on other lags. Default is 0.5.

> 'a_bar_3' : Hyperparamter on exogenous variables. Default is 100.

> 'ARcoeff' : Hyperparameter on first lag coefficient of each equation. Default is 0.9.

> 'coeff' : A N x 2 cell array with specific priors on some coefficients. In the first column you must provide the name of the coefficient, name of dependent + _ + name of rhs variable

```

+ lag specifier. E.g. 'Var1_Var1_lag1' or
'Var1_Var2'. In the last example Var2 is an
exogenous variable. In the second column
you give the prior value, as a scalar double.

> 'method'   : Sets the method to use to draw from the
posterior. Either 'default' (default) or any
other string. 'default' uses (a fixed)
covariance matrix of the shocks (i.e. the
prior). Otherwise it also samples from the
posterior distribution of the covariance
matrix, as in the same way as in the case
of the independent normal wishart (inwishart)
prior. The 'burn', 'thin' and 'S_scale' options
only applies to this last case.

> 'burn'     : How many draws that should be used as burn
in. Default is 500.

> 'thin'      : Every k draws are kept when doing gibbs.
This options sets k. Default is 2. Increase
this option to prevent autocorrelated
draws. See nb_model_generic.checkPosteriorors
to test for autocorrelated draws.

> 'S_scale'   : Prior scale of prior on sigma. Default is 1,
i.e. OLS.

- 'minnesotaMF' : Same as 'minnesota'. This prior option
supports missing observations. Cannot apply
block exogenous variables with this prior.

- 'nwishart'    : Normal-Wishart prior options.

    > 'V_scale'   : Scale of the prior of the variance of the
coefficients. Default is 10.

    > 'S_scale'   : Prior scale of sigma. Default is 1.

- 'nwishartMF'  : Same as 'nwishart', but as we now need a
gibbs sampler the below options are added.
This prior option supports missing
observations. Cannot apply block exogenous
variables with this prior.

    > 'burn'      : How many draws that should be used as burn
in. Default is 500.

    > 'thin'       : Every k draws are kept when doing gibbs.
This options sets k. Default is 2. Increase
this option to prevent autocorrelated
draws. See nb_model_generic.checkPosteriorors
to test for autocorrelated draws.

- 'inwishart'   : Independent Normal-Wishart prior options.

    > 'V_scale'   : Scale of the prior of the variance of the
coefficients. Default is 10.

```

```

> 'S_scale' : Prior scale of sigma. Default is 1.

> 'burn'      : How many draws that should be used as burn
in. Default is 500.

> 'thin'      : Every k draws are kept when doing gibbs.
This options sets k. Default is 2. Increase
this option to prevent autocorrelated
draws. See nb_model_generic.checkPosterior
to test for autocorrelated draws.

- 'inwishartMF' : Same as 'inwishart'. This prior option
supports missing observations.

- For the 'glp', 'glpMF', 'jeffrey', 'minnesota', 'minnesotaMF',
'nwishart' and 'nwishartMF' priors you also have the
oppurtunity to apply the prior for the long run as in
Giannone et. al (2014).

> 'LR'        : Apply priors for the long run. See also
nb_var.applyLongRunPriors. true or false.

> 'phi'       : Shrinkage parameter for the long run prior.
For more see nb_var.applyLongRunPriors.

> 'H'         : Matrix that specifies the cointegration
relations. For more see
nb_var.applyLongRunPriors.

- For the 'glp', 'glpMF', 'jeffrey', 'minnesota', 'minnesotaMF',
'nwishart' and 'nwishartMF' priors you also have the
oppurtunity to apply the sum-of-coefficients prior by Doan,
Litterman, and Sims (1984).

> 'SC'        : Apply sum-of-coefficients prior. true or false.

> 'mu'        : Shrinkage parameter of the sum-of-coefficients
prior. As mu goes to 0 implies the presence
of a unit root in each equation and rules out
cointegration.

- For the 'glp', 'glpMF', 'jeffrey', 'minnesota', 'minnesotaMF',
'nwishart' and 'nwishartMF' priors you also have the
oppurtunity to apply the dummy-initial-observation prior by
Sims (1993).

> 'DIO'       : Apply dummy-initial-observation prior. true or
false.

> 'delta'     : Shrinkage parameter of the dummy-initial-
observation prior. As delta goes to 0 all the
variables of the VAR are forced to be at
their unconditional mean, or the system is
characterized by the presence of an
unspecified number of unit roots without
drift. As such, the dummy-initial observation
prior is consistent with cointegration

- For the 'glp', 'glpMF', 'jeffrey', 'minnesota', 'minnesotaMF',

```

'nwishart' and 'nwishartMF' priors you also have the opportunity to apply the stochastic-volatility-dummy prior "How to estimate a vector autoregression after March 2020" by Lenza and Primiceri (2020).

```

> 'SVD'           : Apply stochastic-volatility-dummy prior.
                     true or false.

> 'periodsMax'  : Number of periods we allow for estimation
                     of s_t. Default is 3.

> 'rho'          : Decay parameter, used for s_t after reaching
                     periodsMax. Default is 0.5.

> 'dateSVD'      : A one line char for where to start the
                     stochastic-volatility-dummy prior. E.g.
                     '2020Q1'. This must be set, if SVD == true!

- 'kkse'          : This is the prior used in the paper by
                     Koop and Korobilis (2014) extended by Schroder
                     and Eraslan (2021) to handle mixed frequency,
                     but adapted to the VAR setting.

> 'f0VarScale'   : Scale factor on the variance of the
                     prior on the initial value of factors.
                     Default is 10. N(0,f0VarScale*I)

> 'lambda0VarScale' : Scale factor on the variance of the
                     prior on the initial value of the factor
                     loadings. Default is 1.
                     N(0,lambdaVarScale*I). !!Remove!!

> 'V0VarScale'    : Scale factor on the mean of the
                     prior on the initial value of the
                     measurement equation covariance matrix.
                     Default is 0.1. Dogmatic prior set to
                     V0VarScale*I. !!Remove!!

> 'Q0VarScale'    : Scale factor on the mean of the
                     prior on the initial value of the
                     state equation covariance matrix.
                     Default is 0.1. Dogmatic prior set to
                     Q0VarScale*I.

> 'gamma'         : Hyperparameter on prior variance of the
                     coefficients of the state equations.
                     On the form V(i,j) = gamma./
                     (ceil(j/options.nFactors).^2). Where
                     V is a matrix with size option.nFactors
                     x option.nLags*option.nFactors. Default
                     value is 0.1.

> 'l_1m'          : Starting value of:
                     Decay factor for the measurement error
                     variance of the monthly variables. A
                     smaller value puts smaller weight on
                     past observations and thus allows for
                     faster parameter change. A value of 1
                     implies constant parameters. Default

```

```

is 1. Do not adjust!

> 'l_1q'      : Starting value of:  

Decay factor for the measurement error  

variance of the quarterly variables. A  

smaller value puts smaller weight on  

past observations and thus allows for  

faster parameter change. A value of 1  

implies constant parameters. Default  

is 1. Do not adjust!

> 'l_2'        : Starting value of:  

Decay factor for the factor error  

variance. A smaller value puts smaller  

weight on past observations and thus  

allows for faster parameter change. A  

value of 1 implies constant parameters.  

Default is 1.

> 'l_3'        : Starting value of:  

Decay factor for the loadings' error  

variance. A smaller value puts smaller  

weight on past observations and thus  

allows for faster parameter change. A  

value of 1 implies constant parameters.  

Default is 1. Do not adjust!

> 'l_4'        : Starting value of:  

Decay factor for the factor VAR  

parameters' error variance.  

A smaller value puts smaller  

weight on past observations and thus  

allows for faster parameter change. A  

value of 1 implies constant parameters.  

Default is 1.

> 'l_1_endo_update' : Controls the endogenous forgetting  

factors  

1: l_1m and l_1q are time-varying/  

endogenous  

0: l_1m and l_1q are constant/static  

Default is 0. Do not adjust!

> 'l_2_endo_update' : Controls the endogenous forgetting  

factors  

1: l_2 is time-varying/endogenous  

0: l_2 is constant/static  

Default is 0.

> 'l_3_endo_update' : Controls the endogenous forgetting  

factors  

1: l_3 is time-varying/endogenous  

0: l_3 is constant/static  

Default is 0. Do not adjust!

> 'l_4_endo_update' : Controls the endogenous forgetting  

factors  

1: l_4 is time-varying/endogenous  

0: l_4 is constant/static

```

```

Default is 0.

- 'dsge' : DSGE-VAR prior. See Del Negro and Schorfheide
(2004), "Priors from General Equilibrium Models for
VARs". Let N be number of dependent variables, L the
number of lags of the VAR, and C is equal to 1 if a
constant is included. otherwise 0.

> 'lambda' : DSGE-VAR weight.

> 'GammaYY' : A N x N double with the nonstandardized sample
moments of the left-hand variables of the VAR.

> 'GammaXX' : A (C + N*L) x (C + N*L) double with the
nonstandardized sample moments of the right-hand
variables of the VAR.

> 'GammaXY' : A (C + N*L) x N double with the nonstandardized
sample moments between the right-hand
variables and the left-hand variables of the
VAR.

```

The last 3 fields can be found from a nb_model_generic model using the method nb_model_generic.getDSGEVARPriorMoments.

Settings for econometric bayesian and hyper-learning:

```

> 'logTransformation' : Set to true to use the log
transformation -log((MAX-VALUE)
./ (VALUE-MIN)) to the parameters to
optimize over. May be important if the
optimizer chosen do not support lower
and upper bounds!

> 'optParam' : A cellstr with the parameters to
optimize over or empty. If empty
it tries to optimize over all hyper
parameters of the selected prior.

```

Settings for missing observations priors:

```

> 'nonStationary' : Set to true to use some tricks to be able
to handle problems related to models in
levels.

- num : Number of prior templates to make.

```

Output:

- options : A struct.

See also:

[nb_var](#), [nb_model_generic.checkPosteriors](#)
[nb_model_generic.getDSGEVARPriorMoments](#)

Written by Kenneth Sæterhagen Paulsen

► **removeObservations** ↑

```
obj = removeObservations(obj,numPer)
```

Description:

Remove the number of wanted observation from each series stored in the data of the object. If one variable contain less observation than another it will still be removed another observation, so as to preserve the ragged edge of the data.

Input:

- obj : A nb_modelData object
- numPer : The amount of observations to remove at the end of the sample for each series. By removal we mean setting it to nan.

Output:

- obj : A nb_modelData object

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODELDATA

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties or fields of the options property of the nb_model_estimate objects (or objects which is of a subclass of this class).

Caution : It may also set the fields of the options property of the object. See the <class>.template and <class>.help for more on the different fields of the options property. E.g. nb_var.template and nb_var.help.

Caution : If the model is reformulated the estimation results and solution will be deleted.

Input:

- obj : A vector of nb_model_estimate objects.

Optional input:

If number of inputs equals 1:

- varargin{1} : A structure of fields to be set. See the template method of each model class for more.

Else:

- varargin : ..., 'inputName', inputValue, ... arguments.

Where you can set all fields of some properties of the object. (options, endogenous, exogenous)

Output:

- obj : A vector of nb_model_estimate objects.

See also:

[nb_model_generic](#), [nb_judgemental_forecast](#), [nb_model_convert](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **setMeasurementEqRestriction** ↑

obj = setMeasurementEqRestriction(obj, restrictions)

Description:

Add restrictions in the measurement equation of the model on the form;

restricted = parameters*variables'

e.g.

X = [0.5, 0.5]*[Y; Z]

where X is a observed variable, while Y and Z are state variables.
So for a mixed frequency var, X and Y must be included in the state equation. An error is provided if this is not the case!

For the xxxMF priors this information is taken into account, otherwise it will only be used during conditional forecasting, i.e. the measurement restrictions will be appended to the solution after estimation is done.

See nb_var.priorTemplate for more on the supported priors.

Input:

- obj : An object of class nb_var

- restrictions : A struct array with fields
> 'restricted' : A one line char with the variable to

restrict. It must be a dependent variable or block exogenous variable.

- > 'parameters' : A cellstr with the names of the variables storing the parameters of the restriction, or a double array. Must have same length as 'variables'.
- > 'variables' : A cellstr with the names of the variables included in the restriction.
- > 'frequency' : Set to the frequency of the observables. This is only important for the mixed frequency VAR model. Give [], if the restriction is on the same frequency as the data. Either 1 (yearly), 4 (quarterly) or 12 (monthly).
- > 'mapping' : Sets the mapping to use for this restricted variable. See the method nb_mfvar.setMapping for more on the mappings to choose from.
- > 'R_scale' : Inverse prior scale of the variance of the measurement error of this restrictions.

The variance of the measurement error is constructed by dividing the variance of each 'restricted' variable by this scaling parameter.

Output:

- obj : An object of class nb_var where the measurement equation restrictions are added.

See also:

[nb_var.template](#), [nb_bVarEstimator.applyMeasurementEqRestriction](#)
[nb_var.priorTemplate](#), [nb_mfvar.setMapping](#)

Written by Kenneth Åkerhagen Paulsen

► **setPrior** ↑

obj = setPrior(obj,prior)

Description:

Set priors of a vector of nb_var objects. The prior input must either be a struct or nb_struct with same size as obj or have size 1.

The provided struct will be added to the property options as the field prior.

See [nb_var.priorTemplate](#) for the format of the struct.

```
Caution : The estim_method field of the option property will  
automatically be set to 'bvar'.
```

```
Caution : Not yet supported for the subclass nb_favar!
```

Input:

- obj : A vector of nb_var objects
- prior : A struct or nb_struct with either matching numel of obj or size
1. If size is 1 all models gets the same prior settings.

Output:

- obj : A vector of nb_var objects with the priors set.

See also:

[nb_var.priorTemplate](#)

Written by Kenneth Sæterhagen Paulsen

► **set_identification ↑**

```
obj = set_identification(obj,type,varargin)
```

Description:

Identify a VAR model. Either using cholesky restrictions or combinations of sign and zero restrictions (short, mid and long term).

The method solve must be called after adding the identifying restrictions

See Andrew Binning (2013), "Underidentified SVAR models: A framework for combining short and long-run restrictions with sign-restrictions". This code is an adaption of his code to the NBToolbox, except that also magnitude restriction is possible using this code.

Input:

- obj : A vector of nb_var objects, or subclasses of this class.
- type : Either 'cholesky' or 'combination'
- varargin (Optional inputs) :
 - > 'cholesky' :
 - 'ordering' : A cellstr with the ordering of the identification.
Default is to use the ordering of the dependent variables of the model. The ordering is so that the

first variable is the variable that is only influenced by one shock in period 1, the second variable is the variable that is influenced by two shock in period 1, and so on.

Caution : All variables of the VAR must be included in this option.

> 'combination' :

- 'maxDraws' : The number of maximal rotations when looking for a identification satifying the sign restrictions. Can be set to inf. Default is 10000.

- 'draws' : The number of wanted draws of the matrix of the map from structural shocks to dependent variables (C). Default is one.

Caution : When it comes to producing IRFs with sign restrictions you have two options. The first is to set 'draws' to a large number, say 1000, and then produce irf for each of those draws. The second approach is to draw parameters using bootstrap, MC methods or from the posterior, and then to make one draw of the C for each draw of the paramters.

- 'restrictions' : A nRestriction x 4 cell;

{Dep, Shock, period, type, value; ...}

> Dep : The dependent variable to add the restriction to. As a string.

> Shock : The name of the identified structural shock. As a string.

> period : The period of the restriction, as a double. For on impact set 0 and for cumulative restriction set it to inf.

Use s:e to add a restriction on the cumulative contribution of a shock to a variable starting at s and ending at e. E.g. 0:10.

> type : Either 0 for zero restriction, '+/' '-' for sign restrictions or '>'/'<' for magnitude restrictions.

Or 'none' : If you want to add a structural shock with no restriction, as you already have exactly identified N-1 shocks.

> value : A 1x1 double with the value of the

magnitude restriction.

Output:

- obj : Identified nb_var objects. See the property field identification of the property solution.

Examples:

See ...\\Econometrics\\test\\test_nb_var

Written by Kenneth Sæterhagen Paulsen
Subfunctions are written by Andrew Binning 2013

► **shock_decomposition ↑**

```
[decomp,decompBand,plotter] = shock_decomposition(obj,varargin)
```

Description:

Shock decomposition of a vector of nb_model_generic objects.

Caution : For recursively estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'packages' : Cell matrix with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

Caution: Two special identifiers can be used; 'Initial Conditions' and 'Steady-state'. The first represent the impact of all shocks that hit the system before the decomposition/estimation started. The second summarizes the impact of steady-state changes on the system, i.e. the impact of break points.

If empty no packing will be done.

- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.
- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'endDate' : End date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create error bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'replic' : The number of simulation for posterior, bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'parallel' : Give true if you want to do the shock decomp in parallel. Default is false.
- 'fcstDB' : An object of class nb_ts with the forecast to decompose. must contain all model variables and shocks/residuals, and must start the period after the estimation/filtering end date or at the estimation/filtering start date. See the nb_model_generic.getForecast method.
- 'type' : Choose 'updated' or 'smoothed'. 'smoothed' is default.
- 'anticipationStartDate' : Start date of anticipating shocks. As a string. If different models has different frequency this date will be converted.
- 'model2' : A struct with the solution of the second model to use for decomposition.
- 'secondModelStartDate' : Start date of second model. As a string.

If different models has different frequency
this date will be converted.

Output:

- decomp : A structure of nb_ts objects with the shock decomposition for each model.
- decompBand : A nested structure of nb_ts objects with the uncertainty bounds of the shock decomposition for each model at each percentile.
- plotter : A 1 x nModel vector of nb_graph_ts objects. Use the graph method to produce the graphs or use the nb_graphPagesGUI on each element of the graph objects.

See also:

[nb_var.parameterDraws](#), [nb_shockDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulate ↑**

```
out = simulate(obj,nSteps,varargin)
```

Description:

Make simulations from model.

Caution : Will only simulate using the full sample estimate of
recursively estimated model.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of simulation steps. As a 1x1 double. Default is 100.

Optional inputs:

- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'bounds' : A struct. Each fieldname must be the name of the variable to add bounds on. Each field must again be a struct with the following fields:
 - 'shock' : Name of the shock to match the restriction on the bounds of the given

variable.

- 'lower' : The lower bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.
- 'upper' : The upper bound of the selected variable. Either a 1x1 double value or a function handle to a probability distribution to draw from.

Caution : The function handles must take one input. I.e. the vector of current observations of the endogenous variables, as a nVar x nSteps double matrix. For the order of the variables see; obj.solution.endo. The value return by the function must be either a 1x1 double or a nSteps x 1 double.

This input can be used to use anticipated shocks to restrict variables to inside a range. E.g. effective lower bound on the interest rate.

- 'condDB' : A nb_ts object with the conditional information to base the simulation upon.
 - 'draws' : Number of residual/innovation draws per parameter draw. Default is 1000. Must be larger than 0.
 - 'foundReplic' : A struct with size 1 x parameterDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.
- Caution: You still need to set 'parameterDraws' to the number of draws you want to do.
- 'method' : The selected method to create density forecast. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input.
 - 'newDraws' : When out of parameter draws, this factor decides how many new draws are going to be made. Default is 0.1. I.e. 1/10 of the parameterDraws input.
 - 'observables' : A cellstr with the observables you want the simulation of. Only for factor models. Will be discarded for all other types of models.
 - 'output' : Either 'endo' (default), 'fullendo' or 'all'. This input indicates which variables to return the simulation of.
 - > 'endo' : All the endogenous variables are returned.

```

> 'fullendo' : All the endogenous variables are
   returned included the lag variables.

> 'full'      : All the variables are returned
   included the lag variables. Also
   including exogenous and shocks.

> 'all'       : All variables are returned (but not
   the lags). Also including exogenous
   and shocks.

- 'parallel'   : Give true if you want to do the simulations in
   parallel. I.e. spread models to different threads.
   Default is false.

- 'parameterDraws' : Number of draws of the parameters. When set to 1
   it will discard parameter uncertainty. Default is 1.

- 'regime'     : Select the regime you want to simulate. If empty
   the simulation will switch between regimes. Only
   an option for Markov switching models.

- 'regimeDraws' : Number of drawn regime paths when doing simulations.
   This will have no effect if the states input is
   providing the full path of regimes. Default is 1.

- 'seed'        : Set simulation seed. Default is 2.0719e+05.

- 'startDate'   : Give the start date of the simulation. If not
   provided the output will be a nb_data object
   without a spesific start date (default). If
   provided the output will be an object of class
   nb_ts.

Caution : If dealing with break-points or exogenous
   time-varying parameters this input must be
   provided.

- 'startingValues' : Either a double or a char:

    > double          : A 1 x nVar double.

    > 'mean'          : Start from the mean of the data
   observations. Default for all
   model except nb_dsge models.

    > 'steadystate'  : Start from the steady state. For
   now only an option for nb_dsge
   models. If you are dealing with a
   MS-model you can indicate the
   state by 'steadystate(state)'.
   E.g. to start in state 1 give;
   'steadystate(1)'. Default for
   nb_dsge models.

    > 'zero'          : Start from zero on all variables
   (Except for the deterministic
   exogenous variables)

```

- 'stabilityTest' : Give true if you want to discard the parameter draws that give rise to an unstable model. Default is false.
- 'startingProb' : Either a double or a char:
 - > double : A nPeriods x nRegime or a 1 x nRegime double. nPeriods refer to the number of recursive periods to forecast.
 - > 'ergodic' : Start from the ergodic transition probabilities. Default.

Output:

- out : A struct with the simulated variables from each model as separate fields. Each field consist of a nSteps x nVar x parameterDraws*draws*regimeDraws nb_data (or nb_ts if the 'startDate' input is used) object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **simulateFromDensity ↑**

```
obj = nb_model_forecast.simulateFromDensity(forecastOutput, draws)
```

Description:

Simulates from the estimated kernel density.

Input:

- forecastOutput : A forecast property of the nb_model_forecast class where the forecastOutput.evaluation stores the kernel density estimate.
- draws : Number of draws to use for simulation.

Output:

- forecastOutput : A struct on the same format as the nb_model_forecast.forecastOutput property

See also:

[nb_model_generic.forecast](#), [forecastPerc2Dist](#)

Written by Per Bjarne Bye and Atle Loneland

Inherited from superclass NB_MODEL_FORECAST

► simulatedMoments ↑

```
[m,c] = simulatedMoments(obj,varargin)
[m,c,ac1] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2] = simulatedMoments(obj,varargin)
[m,c,ac1,ac2,...] = simulatedMoments(obj,varargin)
```

Description:

Calculate simulated moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.
- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.
- 'nLags' : Number of lags to compute when 'stacked' is set to true.
- 'vars' : A cellstr, with a subset of dependent variables of the model. If empty (default), all dependent variables are returned.
- 'type' : Either 'covariance' or 'correlation'. Default is 'correlation'.
- 'draws' : Number of simulations per parameter draw. Default is 1000. Must at least be 1.
- 'pDraws' : Number of parameter draws. Default is 1. I.e. to use the estimate parameters.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return all draws.

Caution: 'pDraws' or 'draws' must be set to a number > 1.
- 'nSteps' : Number of simulation steps. As a 1x1 double. Default is 100.
- 'burn' : The number of periods to remove at start of the simulations. This is to randomize the starting values of the simulation. Default is 0.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more

this input.

- 'demean' : true (demean simulation during estimation of the autocovariance matrix), false (do not). Default is true.
- 'foundReplic' : A struct with size 1 x pDraws. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters. See the parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws' to the number of draws you want to do.

Output:

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.
- varargout{2} : The contemporaneous covariances/correlations, as a nVar x nVar nb_cs object or double. The variance is along the diagonal. (Will be symmetric)
- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar x nVar nb_cs object or double. Along the diagonal is the auto-covariance/correlation with the variable itself. In the upper triangular part the you can find cov(x,y(-i)), where x is the variable along the first dimension. In the lower triangular part the you can find cov(x(-i),y), where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find cov(x,y(-i)) you can use getVariable(varargin{i}, 'y', 'x'), while to get cov(x(-i),y) (== cov(x,y(+i))) you can use getVariable(varargin{i}, 'x', 'y'). (This example only works if the output is of class nb_cs)

See also:

[nb_var.graphCorrelation](#), [nb_var.parameterDraws](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **solve ↑**

obj = solve(obj)

Description:

Solve estimated model(s) represented by nb_var object(s).

Input:

- obj : A vector of nb_var objects.

Output:

- obj : A vector of nb_var objects, where the solved model(s) is/are stored in the property solution.

See also:

[nb_model_generic.solveVector](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **solveNormal** ↑

```
tempSol = nb_var.solveNormal(results,opt,ident)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveRecursive** ↑

```
tempSol = nb_var.solveRecursive(results,opt,ident)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **solveVector** ↑

```
obj = solveVector(obj)
```

Description:

Solve vector of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.

Output:

- obj : A vector of nb_model_generic objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **stateSpace** ↑

```
[x0,P0,d,H,R,T,c,A,B,Q,G,obs,failed] = nb_var.stateSpace(par,nDep,...  
nLags,nExo,restr,restrVal,stabilityTest)
```

Description:

Returns a state-space representation of an VARX(nLags) model.

Observation equation:

$$y = d + Hx + Tz + v$$

State equation:

$$x = c + Ax_{-1} + Gz + Bu$$

Input:

- par : The parameter vector of the VARX model.
 - Order:
 - constant
 - exogenous
 - lagged endogenous
 - residual std
- nDep : The number of dependent variables of the model.
- nLags : the number of lags of the VAR.
- constant : Include constant or not. true or false.
- nExo : Number of exogenous variables included in the model.
- restr : A nDep x nDep*nLags logical matrix. true for the non-restricted parameters.
- restrVal : Values of the restricted parameters.
- stabilityTest : Check stability of system. Sets failed to true if model is not stationary. Default is false.

Output:

- x0 : Initial state vector.
- P0 : Initial variance.
- See equation above
- obs : Index of observables in the state vector.
- failed : See stabilityTest input.

Examples:

See also:

[nb_kalmanlikelihood](#), [nb_kalmanlikelihood_missing](#)

Written by Kenneth Sæterhagen Paulsen

► **struct** ↑

s = struct(obj)

Description:

Convert object to struct.

Input:

- obj : An object of class nb_model_estimate.

Output:

- s : A struct representing the nb_model_generic object.

See also:

[nb_var.unstruct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► **template** ↑

options = nb_var.template()
options = nb_var.template(num)

Description:

Construct a struct which can be provided to the nb_var class constructor.

This structure provided the user the possibility to set different estimation options.

Input:

- num : Number of models to create.

Output:

- options : A struct.

See also:

[nb_var](#), [nb_var.priorTemplate](#)

Written by Kenneth Sætherhagen Paulsen

► **testForecastRestrictions** ↑

probability = testForecastRestrictions(model, restrictions)

Description:

Calculate probability of passing all forecast restrictions

Caution: Nowcast can not yet be tested!

Input:

- obj : A nb_model_forecast object with density forecast
- restrictions : Restrictions as a n x 3 cell array,
{variable, date, test}.
 - > variable : The variable(s) to test,
as a string or cell array of strings.
 - > date : The date as a string. If a cell array is given, a
copy of the restriction will be made for every given
date.
 - > test : Restriction test as a function handle. The value(s)
of the variable(s) in 'variable' at time 'date' are
passed as arguments. Should return a logical.

Output:

- out : A double with the probability

Example:

```
restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...  
{'QSA_URR', 'QSA_DPK_CP'}, '2014Q1', @(x, y) x < y};  
probability = model.testForecastRestrictions(restrictions);
```

See also [nb_testForecastRestrictions](#)

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_FORECAST

► **testParameters** ↑

```
dist = testParameters(obj, expression, draws, method)
```

Description:

Test (non-)linear expression by using bootstrapping (classical) or posterior draws.

Caution (classical only): This method uses that stdBeta is known.
This is the same as using that the t-statistic is normally distributed when stdBeta is known.

Hint : The parameter name can be found by
obj.parameters.name!

Input:

- obj : A scalar solved nb_model_generic object.
- expression : A MATLAB expression as a string. Use parameter names as variables. See model.parameters.name.
- method : A string with the method to use. For bootstrap method see nb_bootstrap. For bayesian models 'posterior' is the only option. Default is 'bootstrap' for classical models, while 'posterior' is the default for bayesian models.
- draws : Number of draws from the parameter distribution. Default is 1000.
- alpha : Confidence level. Default is 0.05.
- type : Type of test:
 - > '=' : Two sided test. expression == 0 (default)
For two-sided tests it is assumed that the distribution is symmetric! Use confidenc/probabillty intervals instead.
 - > '>' : One sided test. expression > 0
 - > '<' : One sided test. expression < 0

Output:

- pval : > 'classical' : P-value of test.
> 'bayesian' : Probabilty of test.

A 1x1 double.
- ci : A 1 x 2 double with the lower and upper bound of the confidence/probabillty interval of the tested expression.
- dist : A nb_distribution object storing the distribution of the tested expression.

Example:

```
dist = obj.testParameters('Parameter1 - Parameter2');
```

Written by Henrik Halvorsen Hortemo

Inherited from superclass NB_MODEL_GENERIC

► theoreticalMoments ↑

```
[m,c]           = theoreticalMoments(obj,varargin)
[m,c,ac1]       = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2]   = theoreticalMoments(obj,varargin)
[m,c,ac1,ac2,...] = theoreticalMoments(obj,varargin)
```

Description:

Calculate theoretical moments; I.e. mean, covariance/correlation, autocovariance/autocorrelation.

Caution : For recursively estimated model only the full sample estimation results is used.

Caution : Only supported for models that can be solved in the following way: $y(t) = A*y(t-1) + B*x(t) + C*e(t)$.

Input:

- obj : An object of class nb_model_generic.

Optional inputs;

- 'vars' : Either 'dependent' or 'full'. 'dependent' will return the moment of the dependent variables of the model without lag, while full will include all lags as well. 'dependent' is default.

Can also be a cellstr, with a subset of variables from 'full'.

- 'maxIter' : Maximum number of iterations to solve the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'tol' : The tolerance level when solving the lyapunov equation. Needed to calculate the covariance matrix of the model.

- 'output' : Either 'nb_cs' or 'double'. Default is 'nb_cs'.

- 'stacked' : If the output should be stacked in one matrix or not. true or false. Default is false.

- 'nLags' : Number of lags to compute when 'stacked' is set to true.

- 'type' : Either 'covariance' or 'correlation'. Default is

```

'correlation'.

- 'pDraws'      : Number of parameter draws. Default is 1. I.e. to use
                   the estimate parameters.

- 'perc'        : Error band percentiles. As a 1 x numErrorBand double.
                   E.g. [0.3,0.5,0.7,0.9]. Default is []. I.e. return the
                   all draws.

Caution: 'pDraws' must be set to a number > 1.

- 'method'       : The selected method to create confidenc/probability
                   bands. Default is '''. See help on the method input of
                   the nb_model_generic.parameterDraws method for more
                   this input.

- 'foundReplic'  : A struct with size 1 x pDraws. Each element
                   must be on the same format as obj.solution. I.e.
                   each element must be the solution of the model
                   given a set of drawn parameters. See the
                   parameterDraws method of the nb_model_generic class.

Caution: You still need to set 'parameterDraws'
          to the number of draws you want to do.

```

Output:

```

- varargout{1} : The mean, as a 1 x nVar x draws nb_cs object or double.

- varargout{2} : The contemporaneous covariances/correlations, as a nVar
                  x nVar nb_cs object or double. The variance is along
                  the diagonal. (Will be symmetric)

- varargout{i} : i > 2. The auto-covariances/correlations, as a nVar
                  x nVar nb_cs object or double. Along the diagonal is the
                  auto-covariance/correlation with the variable itself. In
                  the upper triangular part the you can find cov(x,y(-i)),
                  where x is the variable along the first dimension. In
                  the lower triangular part the you can find cov(x(-i),y),
                  where x is the variable along the first dimension.

E.g: You have two variables x and y, then to find
      cov(x,y(-i)) you can use
      getVariable(varargout{i}, 'y', 'x'),
      while to get cov(x(-i),y) (== cov(x,y(+i))) you
      can use getVariable(varargout{i}, 'x', 'y'). (This
      example only works if the output is of class nb_cs)

```

See also:

[nb_var.graphCorrelation](#), [nb_var.parameterDraws](#)

Written by Kenneth Åkerhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncertaintyDecomposition ↑**

```
[data,plotter] = uncertaintyDecomposition(obj,varargin)
```

Description:

Decompose the density forecast into shocks uncertainty.

Caution : Parameter uncertainty is not yet supported!

Input:

- obj : A vector of nb_model_generic objects
- 'compare2' : An object of class nb_model_generic. If this option is given the decomposition will be done of the difference in the uncertainty between the models. Only an options for 'method' set to 'percentiles'.
- 'method' : One of:
 - > 'percentiles' : This method simulate forecast based on each shock alone, and then calculate the percentile of the variables of interest based on these simulations. Percentiles are not a linear concept, so when aggregating the percentiles of each shock will not add up to the percentile of the distribution of the variable of interest. Therefore the contributions of each residual/shock is scaled with a factor to sum to the the percentiles of this distribution (default).
 - > 'fev' : Calculate the forecast error variances contributed by each shock based on the simulated densities of the residuals/shocks of models forecast.
- 'perc' : At which percentile you want to decompose. Must be scalar double. Default is 0.9. Only an options for 'method' set to 'percentiles'.
- 'packages' : Cell structure with groups of shocks and the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Caution : Not yet supported!

- 'startDate' : Start date of the decomposition. As a string. If different models has different frequency this date will be converted.
- 'variables' : The variables to decompose. Default is the dependent variables defined by the first model.

Output:

- data : Depend on the 'method' input:
 - > 'percentiles' : A nHor x (nExo + nRes)*2 x nVars nb_ts object storing the uncertainty decomposition.
 - > {'fevd','fesd'} : A nHor x nRes x nVars nb_ts object storing the numerically calculated forecast error variances/skewnesses.
- plotter : A nb_graph_ts object which the method graph can be used to produce graphs.

Written by Kenneth Å!terhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **uncondForecast** ↑

obj = uncondForecast(obj,nSteps,varargin)

Description:

Produced unconditional point and density forecast of nb_model_generic objects.

Input:

- obj : A vector of nb_model_generic objects.
- nSteps : Number of forecasting steps. As a 1x1 double. Default is 8.

Optional inputs:

- See the nb_model_generic.forecast method

Output:

- obj : A vector of nb_model_generic objects. See the property forecastOutput.

See also:

[nb_var.forecast](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► [unstruct](#) ↑

```
obj = nb_model_estimate.unstruct(s)
```

Description:

Convert a struct to an object which is a subclass of the nb_model_estimate class. This is how the nb_model_estimate object is loaded.

Input:

- s : A struct. See nb_model_estimate.struct.

Output:

- obj : An object which is a subclass of the nb_model_estimate class.

See also:

[nb_var.struct](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_ESTIMATE

► [update](#) ↑

```
obj      = update(obj,type)
[obj,valid] = update(obj,type,warningOn,inGUI,groupIndex,varargin)
```

Description:

Update data of model, and optionally re-estimate, re-solve and forecast the models given the update data

Input:

- obj : A vector of object of class nb_model_generic
- type : > '' : Only update data. Default
 > 'estimate' : Update data and estimate.
 > 'solve' : Update data, estimate and solve
 > 'forecast' : Update data, estimate, solve and forecast.

- warningOn : 'on' or 'off'. If 'on' only warnings are given while the data source is updated, else an error will be given. Default is 'on'. Set it to 'off', if the 'write' option is used.
- inGUI : 'on' or 'off'. Indicate if the update command is called in the GUI or not. Default is 'off'. When 'on' a waitbar will be created to tell the user about the status.
- groupIndex : A 1 x nLevel double with the group level. Default is [].

Optional input:

- 'write' : Give this string as one of the optional inputs to write an error file if updating of data, estimation or forecasting failes. All objects will be returned, but can be removed by indexing by the valid output.
- 'waitbar' : Give this string as one of the optional inputs to add waitbar for the loop over models in the estimation step.

Output:

- obj : A vector of object of class nb_model_generic
- valid : A logical with size 1 x nObj. true at location ii if estimation or forecasting of model ii succeeded, otherwise false. If 'write' is not used an error will be thrown instead, so in this case this output will be true for all models.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

► **variance_decomposition ↑**

```
[decomp, decompBands, plotter, plotterBands] = ...
    variance_decomposition(obj, varargin)
```

Description:

Variance decomposition of a vector of nb_model_generic objects.

Caution : For recursivly estimated model only the full sample estimation results is used.

Input:

- obj : A vector of nb_model_generic objects
- 'horizon' : The horizons. As a 1 x nHor double. Default is [1:8, inf].

- 'replic' : The number of simulation for bootstrap and MC methods. Default is 1. Only used when 'perc' is provided.

Caution: Will not have anything to say for the method 'identDraws'. See the option 'draws' of the method nb_var.set_identification for more.
- 'shocks' : Which shocks to calculate variance decomposition of. Default is the residuals defined by the first model.
- 'variables' : The variables to create variance decomp of. Default is the dependent variables defined by the first model.
- 'perc' : Error band percentiles. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is not to provide error bands.
- 'method' : The selected method to create confidenc/probability bands. Default is ''. See help on the method input of the nb_model_generic.parameterDraws method for more this input. An extra option is:
 - > 'identDraws' : Uses the draws of the matrix with the map from structural shocks to dependent variables. See nb_var.set_identification. Of course this option only makes sence for VARs identified with sign restrictions.
- 'output' : Either 'closest2median' (default) or 'median'.
 - > 'closest2median' : The median is represented by the model(s) that are closest to the actual median.
 - > 'median' : The median represented by the numerical statistic.

Caution : Only an option when 'replic' > 1 and 'perc' is nonempty.
- 'parallel' : Give true if you want to do the forecast in parallel. Default is false.
- 'regime' : Select the regime to do the FEVD of. Only for Markov-switching model. If empty the ergodic mean will be used.
- 'foundReplic' : A struct with size 1 x replic. Each element must be on the same format as obj.solution. I.e. each element must be the solution of the model given a set of drawn parameters.
- 'stabilityTest' : Give true if you want to discard the draws that give rise to an unstable model. Default is false.
- 'packages' : Cell matrix with groups of shocks and

the names of the groups. If you have not listed a shock it will be grouped in a group called 'Rest'

Must be given in the following format:

```
{'shock_group_name_1',...,'shock_group_name_N';
 'shock_name_1'      ,...,'shock_name_N'}
```

Where 'shock_name_x' must be a string with a shock name or a cellstr array with the shock names. E.g 'E_X_NW' or {'E_X_NW','E_Y_NW'}.

If empty no packing will be done.

Output:

- decomp : A structure of nb_ts objects with the variance decomosition for each model. (If simulated it will store the median or closest to median dependent on the 'output' option)
- decompBands : A structure of structs with the percentiles of the variance decomosition for each model. For each percentiles the output is as decomp.
- plotter : A 1 x nModel vector of nb_graph_cs objects. Use the graph method or the nb_graphPagesGUI class to produce the graphs.
- plotterBands: A 1 x nModel struct. Each field is a 1 x nVars vector of nb_graph_cs objects. Use the graphSubPlots method to produce the graphs of each of the fields. This graphs will plot the contribution of each shock to each variable with error bands.

See also:

[nb_var.parameterDraws](#), [nb_varDecomp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_MODEL_GENERIC

◆ **nb_calcMultiplier**

```
m = nb_calcMultiplier(dx,dy,r,gross)
```

Description:

Calculate multiplier using the formula (3) page 11 of "Clearing Up the Fiscal Multiplier Morass" (2015) by Leeper, Traum and Walker.

Input:

```
- dx      : A nObs x nVar x nSim double with the affected variables.  
- dy      : A nObs x 1 x nSim double with the policy variable of interest.  
- r       : A nObs x 1 x nSim double with the gross or net interest rates.  
- gross   : Give true if r is the gross interest rate (default) or false  
            if it is the net interest rate.
```

Output:

```
- m      : A 1 x nVar x nSim double with the calculated multipliers.
```

See also:

[nb_dsge.calculateMultipliers](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_compareCoeffs**

```
out = nb_compareCoeffs(model1,model2,sort)
```

Description:

Use this function to compare parameters between two models. Inputs can be coefficient files (structs) or nb_dsge-model objects. Only parameters that are found in both models will be displayed.

Input:

```
- model1 : Model 1 (coefficient file (struct) or nb_dsge-model object)  
- model2 : Model 2 (coefficient file (struct) or nb_dsge-model object)  
- sort   : Sorts output by relative difference. Logical. (optional). Default  
            is false (no sorting).
```

Output:

```
out      : Cell matrix that shows parameter names, values and absolute  
            and relative differences.
```

Examples: `out = nb_compareCoeffs(m_old, m_new, TRUE)`

See also:

Written by Erling M. Kravik

◆ **nb_COMPARESteadystate**

```
out = nb_compareCoeffs(model1,model2,sort)
```

Description:

Use this function to compare steady state values between two models. Inputs are nb_dsge-models. Only variables that are found in both models will be displayed.

Input:

- model1 : Model 1 (nb_dsge-model object)
- model2 : Model 2 (nb_dsge-model object)
- sort : Sorts output by relative difference. Logical. (optional). Default is false (no sorting). Only available from MatLab 2017.

Output:

out : Cell matrix that shows variable names, values and absolute and relative differences.

Examples: `out = nb_compareSteadystate(m_old, m_new, TRUE)`

See also:

Written by Erling M. Kravik

◆ **nb_estimateDensityForforecast**

```
[vars,dist,dates] = nb_estimateDensityForforecast(model,start,hor)
```

Description:

Estimate distribution of the density forecast at a given period (date).

Input:

- model : A scalar nb_model_generic or nb_model_group object.
- start : The start date of the forecast of interest. As a string or a nb_date object. Default is the last forecast data.
- hor : Max horizon of interest. Default is as many as the model has forecasted.
- vars : A cellstr with variables of interest. Default is all the variables that have been forecast by the model.

Output:

- dist : A hor x nVars nb_distribution object.
- dates : A hor x 1 cellstr with the dates of the forecast.
- vars : A 1 x nVars cellstr with the variables of the forecast.

Written by Kenneth Sætherhagen Paulsen

◆ nb_forecast2Excel

`nb_forecast2Excel(fcst,filename)`

Description:

Save down forecast return by the `nb_model_vintages.getForecast` method to excel.

Input:

- fcst : An object of class `nb_ts`.
- filename : Name of excel file. E.g. '`test`'.

See also:

[nb_model_vintages.getForecast](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_getModelNames

`names = nb_getModelNames(varargin)`

Description:

Get model names, if some model names are empty, they will be given the default name '`Model<ii>`'.

Input:

- varargin : Each input must be an object of class `nb_model_generic` or `nb_model_group`

Output:

- names : A 1 x N cellstr.

Written by Kenneth Sætherhagen Paulsen

◆ nb_isModelMarkovSwitching

```
ret = nb_isModelMarkovSwitching(model)
```

Description:

Test if a model is a Markov switching. See nb_model_generic.solution for more on the model input.

Output:

- ret : true if Markov switching model.

Written by Kenneth Sæterhagen Paulsen

◆ nb_plotPPAMultipliers

```
plotter = nb_plotPPAMultipliers(mult,shock,variable,perc)
```

Description:

Plot output from the priorPredictiveAnalysis method when the method input is set to calculateMultipliers.

Input:

- mult : The output from priorPredictiveAnalysis.
- shock : The shock you want to graph. As a one line char.
- variable : The variable you want to graph. As a one line char.
- perc : This options sets the error band percentiles of the graph. As a 1 x numErrorBand double. E.g. [0.3,0.5,0.7,0.9]. Default is 0.68.

Output:

- plotter : An object of class nb_graph_cs, use the graph method or the nb_graphPagesGUI class. If nargout == 0, the graph will be made using the nb_graphPagesGUI class.

See also:

[nb_graph_cs](#), [nb_graphPagesGUI](#)

Written by Kenneth Sæterhagen Paulsen

25.19 Optimization

- nb_abc
 - nb_callOptimizer
 - nb_fitness.standardDev
 - nb_fitness.standardDevSqrt
 - nb_getOptimizers
 - nb_optimizerDisplayer
 - nb_pso
 - pso.binary
 - pso.optimset
 - nb_bee
 - nb_fitness.standard
 - nb_fitness.standardDevSq
 - nb_getDefaultOptimset
 - nb_interpretExitFlag
 - nb_optimizerMessageState
 - pso README.md
 - pso.do
-

► nb_fitness.standard ↑

```
fit = nb_fitness.standard(fVal,fMin,Q)
```

Description:

Calculates the fitness function

```
fit = 1/(1 + fVal) if fVal >= 0  
fit = 1 + abs(fVal) if fVal < 0
```

Input:

- fVal : Value of function. As a N x M double.
- fMin : Minimum value of function obtained until now. As a scalar double. Not in use.
- Q : Scaling parameter. Not in use.

Output:

- fit : Fitness score. As a N x M double.

Written by Kenneth Sæterhagen Paulsen

► nb_fitness.standardDev ↑

```
fit = nb_fitness.standardDev(fVal,fMin,Q)
```

Description:

Calculates the fitness function

```
fit = 1/(Q + fVal - fMin)
```

Input:

- fVal : Value of function. As a N x M double.
- fMin : Minimum value of function obtained until now. As a scalar double.
- Q : Scaling parameter.

Output:

- fit : Fitness score. As a N x M double.

Written by Kenneth Sæterhagen Paulsen

► **nb_fitness.standardDevSq** ↑

```
fit = nb_fitness.standardDevSq(fVal, fMin, Q)
```

Description:

Calculates the fitness function

```
fit = 1/(Q + (fVal - fMin)^2 )
```

Input:

- fVal : Value of function. As a N x M double.
- fMin : Minimum value of function obtained until now. As a scalar double.
- Q : Scaling parameter.

Output:

- fit : Fitness score. As a N x M double.

Written by Kenneth Sæterhagen Paulsen

► **nb_fitness.standardDevSqrt** ↑

```
fit = nb_fitness.standardDevSqrt(fVal, fMin, Q)
```

Description:

Calculates the fitness function

```
fit = 1/(Q + sqrt(fVal - fMin) )
```

Input:

- fVal : Value of function. As a N x M double.
- fMin : Minimum value of function obtained until now. As a scalar double.
- Q : Scaling parameter.

Output:

- fit : Fitness score. As a N x M double.

Written by Kenneth Sæterhagen Paulsen

► pso.binary ↑

```
psobinary(fitnessfcn,nvars)
psobinary(fitnessfcn,nvars,options)
```

Description:

Particle swarm optimization for binary genomes.

This function will optimize fitness functions where the variables are row vectors of size 1xnvars consisting of only 0s and 1s.

PSO.BINARY is provided as a wrapper for PSO.DO, to avoid any confusion. This is because the binary optimization scheme is not designed to take any constraints. PSO.BINARY does not allow the passing of constraints. It takes a given optimization problem with binary variables, and automatically sets the options structure so that 'PopulationType' is 'bitstring'.

This has exactly the same effect as setting the appropriate options manually, except that it is not possible to unintentionally define constraints, which would be ignored by the binary variable optimizer anyway.

Problems with hybrid variables (double-precision and bit-string combined) cannot be solved yet.

The output variables for PSO.BINARY is the same as for PSO.DO.

See also:

[nb_pso](#), [pso.do](#), [pso.optimset](#)

Written by S. Samuel Chen. Version 1.31.2.
Available from <http://www.mathworks.com/matlabcentral/fileexchange/25986>
Distributed under BSD license. First published in 2009.

Edited by Kenneth S. Paulsen
- Made it into a function of the pso package. 10/2018

► pso.do ↑

```
x = pso.do(fitnessfcn,nvars)
x = pso.do(fitnessfcn,nvars,Aineq,bineq)
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq)
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,LB,UB)
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,LB,UB,nonlcon)
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,LB,UB,nonlcon,options)
x = pso.do(problem)
[x, fval,exitflag] = pso.do(...)
[x, fval,exitflag,output] = pso.do(...)
[x, fval,exitflag,output,population] = pso.do(...)
[x, fval,exitflag,output,population,scores] = pso.do(...)
```

Description:

Find the minimum of a function using Particle Swarm Optimization.

This is an implementation of Particle Swarm Optimization algorithm using the same syntax as the Genetic Algorithm Toolbox, with some additional options specific to PSO. Allows code-reusability when trying different population-based optimization algorithms. Certain GA-specific parameters such as cross-over and mutation functions will not be applicable to the PSO algorithm. Demo function included, with a small library of test functions. Requires Optimization Toolbox.

New features will be added regularly until this is made redundant by an official MATLAB PSO toolbox.

```
x = pso.do(fitnessfcn,nvars)
Runs the particle swarm algorithm with no constraints and default
options. fitnessfcn is a function handle, nvars is the number of
parameters to be optimized, i.e. the dimensionality of the problem. x is
a 1xnvars vector representing the coordinates of the global optimum
found by the pso algorithm.
```

```
x = pso.do(fitnessfcn,nvars,Aineq,bineq)
Linear constraints, such that Aineq*x <= bineq. Aineq is a matrix of size
nconstraints x nvars, while b is a column vector of length nvars.
```

```
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq)
Linear equality constraints, such that Aeq*x = beq. 'Soft' or 'penalize'
boundaries only. If no inequality constraints exist, set Aineq and bineq
to [].
```

```
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,lb,ub)
Lower and upper bounds defined as LB and UB respectively. Both LB and UB,
if defined, should be 1 x nvars vectors. Use empty arrays [] for Aineq,
```

bineq, Aeq, or beq if no linear constraints exist.

```
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,LB,UB,nonlcon)
Non-linear constraints. Nonlinear inequality constraints in the form c(x)
<= 0 have now been implemented using 'soft' boundaries, or
experimentally, using 'penalize' constraint handling method. See the
Optimization Toolbox documentation for the proper syntax for defining
nonlinear constraints. Use empty arrays [] for Aineq, bineq, Aeq, beq,
LB, or UB if they are not needed.
```

```
x = pso.do(fitnessfcn,nvars,Aineq,bineq,Aeq,beq,LB,UB,nonlcon,options)
Default algorithm parameters replaced with those defined in the options
structure:
Use >> options = psooptimset('Param1','value1','Param2','value2',...
to generate the options structure. Type >> psooptimset with no input or
output arguments to display a list of available options and their
default values.
```

NOTE: the swarm will perform better if the PopInitRange option is defined so that it closely bounds the expected domain of the feasible region. This is automatically done if the problem has both lower and upper bound constraints, but not for linear and nonlinear constraints.

NOTE 2: If options.HybridFcn is to be defined, and if it is necessary to pass a non-default options structure to the hybrid function, then the options structure may be included in a cell array along with the pointer to the hybrid function. Example:

```
>> % Let's say that we want to use fmincon to refine the result from PSO:
>> hybridoptions = optimset(@fmincon) ;
>> options.HybridFcn = {@fmincon, hybridoptions} ;
```

NOTE 3:

Perez and Behdinan (2007) demonstrated that the particle swarm is only stable if the following conditions are satisfied:

Given that C0 = particle inertia

```
C1 = options.SocialAttraction
C2 = options.CognitiveAttraction
1) 0 < (C1 + C2) < 4
2) (C1 + C2)/2 - 1 < C0 < 1
```

If conditions 1 and 2 are satisfied, the system will be guaranteed to converge to a stable equilibrium point. However, whether or not this point is actually the global minimum cannot be guaranteed, and its acceptability as a solution should be verified by the user.

```
x = pso.do(problem)
Parameters imported from problem structure.
```

```
[x,fval] = pso.do(...)
Returns the fitness value of the best solution.
```

```
[x,fval,exitflag] = pso.do(...)
Returns information on outcome of pso run. This should match exitflag
values for GA where applicable, for code reuseability between the two
toolboxes.
```

```
[x,fval,exitflag,output] = pso.do(...)
The structure output contains more detailed information about the PSO
run. It should match output fields of GA, where applicable.
```

```
[x,fval,exitflag,output,population] = pso.do(...)  
A matrix population of size MaxNodes x nvars, with the locations of  
particles across the rows. This may be used to save the final positions  
of all the particles in a swarm.
```

```
[x,fval,exitflag,output,population,scores] = pso.do(...)  
Final scores of the particles in population.
```

Bibliography

J Kennedy, RC Eberhart, YH Shi. Swarm Intelligence. Academic Press, 2001.

SM Mikki, AA Kishk. Particle Swarm Optimization: A Physics-Based Approach. Morgan & Claypool, 2008.

RE Perez and K Behdinan. "Particle swarm approach for structural design optimization." Computers and Structures, Vol. 85:1579-88, 2007.

See also:

[nb_pso](#), [pso.optimset](#)

Written by S. Samuel Chen. Version 1.31.2.
Available from <http://www.mathworks.com/matlabcentral/fileexchange/25986>
Distributed under BSD license. First published in 2009.

Edited by Kenneth S. Paulsen

- Made it into a function of the pso package. 10/2018
- Removed the default problem to solve. 11/2018

► **pso.optimset** ↑

```
pso.optimset  
options = pso.optimset  
options = pso.optimset(@pso)  
options = pso.optimset(@psomultiobj)  
options = pso.optimset('param1',value1,'param2',value2,...)  
options = pso.optimset(olddopts,'param1',value1,...)  
options = pso.optimset(olddopts,newopts)
```

Description:

`psooptimset` with no input or output arguments displays a complete list of parameters with their valid values.

`options = psooptimset` (with no input arguments) creates a structure called `options` that contains the options, or parameters, for the `pso` algorithm and sets parameters to `[]`, indicating default values will be used.

`options = psooptimset(@pso)` creates a structure called `options` that contains the default options for the genetic algorithm.

`options = psooptimset(@psomultiobj)` creates a structure called `options` that contains the default options for `psomultiobj`. Not yet implemented

`options = psooptimset('param1',value1,'param2',value2,...)` creates a structure options and sets the value of 'param1' to value1, 'param2' to value2, and so on. Any unspecified parameters are set to their default values. Case is ignored for parameter names.

`options = psooptimset(olddopts,'param1',value1,...)` creates a copy of oldopts, modifying the specified parameters with the specified values.

`options = psooptimset(olddopts,newopts)` combines an existing options structure, oldopts, with a new options structure, newopts. Any parameters in newopts with nonempty values overwrite the corresponding old parameters in oldopts.

Again, type `psooptimset` with no input arguments to display a list of options which may be set.

NOTE regarding the `ConstrBoundary` option:

The 'penalize' constraint management option as described in Perez 2007 is the default since it seems to provide the best combination of performance and versatility. It combines the best of the 'soft' and 'absorb' methods and adds a penalty calculated for every infeasible point in the search space.

Although the 'penalize' method is preferred, the 'soft' or 'absorb' style boundaries may still be used. They are described below. They are not very effective if your problem contains any nonlinear equality constraints of the form $c(x) = 0$ or linear equality constraints of the form $A_{eq}x = b_{eq}$. If your problem does not require any equality constraints, then calculations may be faster with 'soft' boundaries since infeasible solutions are not evaluated at all and simply assigned a fitness score of infinity.

A 'soft' boundary allows particles to leave problem bounds, but sets their fitness scores to Inf if they do. This can save time, since infeasible points are not evaluated. This will also save you some trouble if your fitness function has the possibility of throwing errors if it tries to evaluate an infeasible point in the search space.

The 'reflect' and 'absorb' options prevent the particles from travelling outside the problem bounds at all. Note that 'reflect' has only been implemented for bounded constraints, and is not really suitable for any other problem type at this time. The 'absorb' method may suffer from poor performance if linear or nonlinear equality constraints are used. Both are likely to be deprecated in the near future.

NOTE regarding cognitive and social attraction parameters:

Perez and Behdinan (2007) demonstrated that the particle swarm is only stable if the following conditions are satisfied:

Given that C_0 = particle inertia

C_1 = `options.SocialAttraction`

C_2 = `options.CognitiveAttraction`

- 1) $0 < (C_1 + C_2) < 4$
- 2) $(C_1 + C_2)/2 - 1 < C_0 < 1$

If conditions 1 and 2 are satisfied, the system will be guaranteed to converge to a stable equilibrium point. However, whether or not this point is actually the global minimum cannot be guaranteed, and its acceptability as a solution should be verified by the user.

Bibliography

RE Perez and K Behdinan. "Particle swarm approach for structural design optimization." Computers and Structures, Vol. 85:1579-88, 2007.

See also:

[nb_pso](#), [pso.do](#)

Written by S. Samuel Chen

Edited by Kenneth S. Paulsen

- Made it into a function of the pso package. 10/2018
- Changed TimeLimit to MaxTime. 10/2018
- Changed Generations to MaxIter. 10/2018
- Changed PopulationSize to MaxNodes. 10/2018
- Changed OutputFcns to OutputFcn. 10/2018
- Removed the PlotFcns and PlotInterval options. 10/2018

■ nb_abc

Go to: [Properties](#) | [Methods](#)

An implementation of the artificial bee colony algorithm by Karaboga.

References:

"A comprehensive survey: Artificial bee colony (ABC) algorithm and applications", Karaboga et. al (2012).

"Modified artificial bee colony algorithm for constrained problems optimization", Stanarevic et. al (2015)

This class tries to solve the following problem

min Q(x)
x in S

Or the constrained problem

min Q(x)
x in S

s.t. x in F

Where Q is the objective to minimize and x is the inputs to the objective. S is the search space, i.e. the space to look for candidate values that minimizes the objective Q. F is the feasible space, and can be provided by using the constraints property. You can apply both inequality constraints $g(x) \leq 0$ or equality constraints $h(x) == 0$. In the latter case you will apply the inequality constraints $\text{abs}(h(x)) - \text{tolerance} \leq 0$, where tolerance is a small number. It can be set using the field tolerance of the options struct.

Caution: The steps of this implementation is slightly different from that of Karaboga et. al (2012).

Initialize employed bees by scouting.

Repeat:

1. Employed bees that has reached the cutLimit are sent out scouting. See nb_bee.scout.
2. Send onlookers to join the employed bees by random. Higher probabilities are assign to the employed bees that communicates good values of the objective. Finally onlookers are sent to explore the neighbourhood of the selected employed bee by changing the value of one parameter, which is selected at random. See nb_bee.joinDance. The probabilities are calculated accordingly to Stanarevic et. al (2015) in the case of constrained minimization.
3. Employed bees are then sent to explore their neighbourhood by changing the value of one parameter, which is selected at random. See nb_bee.dance.
4. All the new points of the bees are evaluated. (This is the step that can be ran in parallel.)
5. The best point in the parameter space is saved.

This is repeated until some of the limits (maxTime, maxFunEvals, maxIter or maxTimeSinceUpdate are reached).

Superclasses:

handle

Constructor:

```
obj = nb_abc(fh,init,lb,ub)
obj = nb_abc(fh,init,lb,ub,options,varargin)
```

Input:

- fh : See the documentation of the objective property of the nb_abc class.
- init : See the documentation of the initialXValue property of the nb_abc class.
- lb : See the documentation of the lowerBound property of the nb_abc class.
- ub : See the documentation of the upperBound property of the nb_abc class.
- options : See the nb_abc.optimset method for more on this input.

Optional inputs:

- varargin : Optional number of inputs given to the objective when it is called during the minimization algorithm.

Output:

- obj : An object of class nb_abc.

See also:

[nb_abc.optimset](#), [nb_abc.call](#), [nb_fmin](#), [fmincon](#), [fminunc](#), [fminsearch](#)

bee_gate, (RISE, toolbox)

Written by Kenneth Sæterhagen Paulsen

Properties:

- bees
- constraints
- cutLimit
- display
- displayer
- doHessian
- employedShare
- exitFlag
- fitnessFunc
- fitnessScale
- funEvals
- hasConstraints
- hessian
- initialXValue
- iterations
- lowerBound
- maxFunEvals
- maxIter
- maxNodes
- maxTime
- maxTimeSinceUpdate
- maxTrials
- minFunctionValue
- minXValue
- numWorkers
- objective
- objectiveLimit
- options
- saveName
- saveTime
- tolerance
- upperBound
- useParallel

- bees ↑

A vector of nb_bee objects.

- constraints ↑

A function handle that applies the linear or non-linear constraint on the parameters. If not given, or given as [], unconstrained minimization will be done.

It must return two outputs. The first output must be a vector with the values of the inequality constraints $g(x) \leq 0$. Positive numbers means violation. The second input must return a vector with the values of the equality constraints $h(x) == 0$. If $\text{abs}(h(x)) > \text{options.tolerance}$ means a violation.

Example:

```
function [c,ceq] = conFunc(x,a)
    c = a*x(1) - x(2); % inequality constraint a*x(1) - x(2) <= 0
    ceq = x(3) - x(2); % equality constraint x(3) - x(2) == 0
end
```

- cutLimit ↑

Positive integer. The maximum trial that is allowed for employed bee before it starts to scout for another area. Default is 20. Must be greater than 4.

- display ↑

'iter' Level of display. 'iter' displays output at each iteration. Only supported option.

- **display** ↑

An object of class nb_optimizerDisplayer. This object set the way the optimizer displayes result during the optimization.

- **doHessian** ↑

Set to false to not calculate the Hessian at the found minimum. If true the Hessian will be calculated using the nb_hessian function. Default is true

- **employedShare** ↑

This set the share of employed bees. Default is 0.33, i.e. 33%. Must be in the set (0.1,0.9).

- **exitFlag** ↑

The reason for exiting. One of:

```
> 0 : Maximum function evaluation limit exceeded (maxFunEvals).  
> -1 : Maximum iteration limit exceeded (maxIter).  
> -2 : Maximum time limit exceeded (maxTime).  
> -3 : Maximum time limit since last update exceeded  
        (maxTimeSinceUpdate).  
> -4 : User stop the optimization manually.
```

- **fitnessFunc** ↑

A function handle that return the fitness of each bee at each iteration of the optimization. See for example nb_fitness.standard, which is the default fitness function. Other proper fitness functions may be found in the nb_fitness package.

- **fitnessScale** ↑

This set the value of the fitness scaling parameter. Default is 1. This is the third input to the function given by the fitnessFunc property.

- **funEvals** ↑

Number of function evaluations.

- **hasConstraints** ↑

true if minimization is constrained, otherwise false. See constraints property.

- **hessian** ↑

A nVar x nVar double with the estimated Hessian at the found minimum.

- **initialXValue** ↑

The initial values of the optimization. Default is zero. As a N x 1 double.

- **iterations** ↑

Number of iteration of the abc algorithm.

- **lowerBound** ↑

The lower bound on the x values. As a N x 1 double. Must be finite!

- **maxFunEvals** ↑

Positive integer. Maximum number of function evaluation allowed. Default is inf.

- **maxIter** ↑

Positive integer. Maximum number of iterations allowed. Default is inf.

- **maxNodes** ↑

Positive integer. The number of bees to use during optimization.

- **maxTime** ↑

Positive integer. The maximum time spent on optimization in seconds.

- **maxTimeSinceUpdate** ↑

Positive integer. The maximum time spent on optimization since last time the minimum point where updated (in seconds).

- **maxTrials** ↑

Positive integer. The maximum trial that is allowed for finding initial candidates. Default is 100.

- **minFunctionValue** ↑

The minimized value found during the optimization. As a scalar double.

- **minXValue** ↑

The x values at the located minimum. As a nVar x 1 double.

- **numWorkers** ↑

Positive integer. Number of worker to use when running optimization in parallel.

- **objective** ↑

The objective to minimize. As a function handle. Must take a N x 1 double as the first input.

- **objectiveLimit** ↑

Double. If the objective return a value greater than this, a new candidate of x is drawn. Default is 1e9.

- **options** ↑

A struct with the options of the optimiziation. See the nb_abc.optimset method for more on this option.

- **saveName** ↑

Name of the file to save the object, with path, but without extension.

- **saveTime** ↑

The time between saving of the object.

- **tolerance** ↑

Tolerance value used for equality constraint optimization problems.

- **upperBound** ↑

The upper bound on the x values. As a n x 1 double. Must be finite!

- **useParallel** ↑

true or false. Give true if the nodes should be spread on different parallel workers.

Methods:

- call • minimize • optimset • restart • set
- test • testConstrained • testParallel • testSave

► **call** ↑

```
[x,fval,exitflag,hessian] = nb_abc.call(fh,init,lb,ub,options,constraints,varargin)
```

Description:

Call the artificial bee colony algorithm. See the documentation of the nb_abc class for more on this algorithm.

Input:

- fh : See the documentation of the objective property of the nb_abc class.
- init : See the documentation of the initialXValue property of the nb_abc class.
- lb : See the documentation of the lowerBound property of the nb_abc class.
- ub : See the documentation of the upperBound property of the nb_abc class.
- options : See the nb_abc.optimset method for more on this input.
- constraints : See the documentation of the constraints property of the nb_abc class.

Optional inputs:

- varargin : Optional number of inputs given to the objective when it is called during the minimization algorithm.

Output:

- x : See the documentation of the minXValue property of the nb_abc class.
- fval : See the documentation of the minFunctionValue property of the nb_abc class.
- exitflag : See the documentation of the exitFlag property of the nb_abc class.
- hessian : See the documentation of the hessian property of the nb_abc class.

Examples:

```
g = @(x)-sin(x);  
x3 = nb_abc.call(g,-3,-4,4)
```

See also:

[nb_abc.optimset](#), [fmincon](#), [fminunc](#), [fminsearch](#), [bee_gate](#)

Written by Kenneth Sæterhagen Paulsen

► **minimize** ↑

```
minimize(obj)
minimize(obj,varargin)
```

Description:

Run minimization on the given objective.

Input:

- obj : An object of class nb_abc.

Optional input:

- varargin : Given to the nb_abc.set method.

Output:

The provided object of class nb_abc has been updated the following properties;
> minXValue
> minFunctionValue
> exitFlag
> hessian

See also:

[nb_abc.call](#)

Written by Kenneth Sæterhagen Paulsen

► **optimset** ↑

```
options = nb_abc.optimset(varargin)
```

Description:

Get optimization settings or help.

Optional input:

- Run without inputs to get defaults.
- Run with 'list' as the first input to get a cellstr with the supported options.
- Run with 'help' to get a struct with help on each option.

- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:

[nb_abc.options](#)

Written by Kenneth Sæterhagen Paulsen

► **restart** ↑

```
[x,fval,exitflag,hessian] = restart(obj)
```

Description:

Call the artificial bee colony algorithm. See the documentation of the nb_abc class for more on this algorithm.

Input:

- obj : A nb_abc object that has been stopped or saved.

Output:

- x : See the documentation of the minXValue property of the nb_abc class.
- fval : See the documentation of the minFunctionValue property of the nb_abc class.
- exitflag : See the documentation of the exitFlag property of the nb_abc class.
- hessian : See the documentation of the hessian property of the nb_abc class.

See also:

[nb_abc.call](#)

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Set properties of the object using 'propertyName', PropertyValue pairs.

Input:

- obj : An object of class nb_abc.

Optional inputs:

- varargin: 'propertyName', PropertyValue pairs

Output:

- obj : An object of class nb_fmin where the wanted properties has been set.

Examples:

```
set(obj,'initialXValue',2,'lowerBound',0)
```

See also:

[nb_abc.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► **test ↑**

```
nb_abc.test()
```

Description:

Test the ABC algorithm on the Rosenbrock Valley function;

```
f = 100*(x(2) - x(1)^2)^2 + (x(1) - 1)^2
```

The optimizer stops when it is 60 second since last time the minimum where updated.

Output:

- [x,fval,exitflag,hessian] : See the nb_abc.call method for more on these outputs.

See also:

[nb_abc.call](#)

Written by Kenneth Sæterhagen Paulsen

► **testConstrained** ↑

`nb_abc.testConstrained()`

Description:

Test the ABC algorithm on the Rosenbrock Valley function;

$$f = 100 * (x(2) - x(1)^2)^2 + (x(1) - 1)^2$$

$$\text{s.t. } x(2) - x(1) \leq 0$$

The optimizer stops when it is 60 second since last time the minimum where updated.

Output:

- `[x,fval,exitflag,hessian]` : See the `nb_abc.call` method for more on these outputs.

See also:

[nb_abc.call](#)

Written by Kenneth Sæterhagen Paulsen

► **testParallel** ↑

`nb_abc.testParallel()`

Description:

Test the ABC algorithm in parallel on the Rosenbrock Valley function;

$$f = 100 * (x(2) - x(1)^2)^2 + (x(1) - 1)^2$$

Output:

- `[x,fval,exitflag,hessian]` : See the `nb_abc.call` method for more on these outputs.

See also:

[nb_abc.call](#)

► **testSave** ↑

`nb_abc.test()`

Description:

Test the ABC algorithm on the Rosenbrock Valley function;

`f = 100*(x(2) - x(1)^2)^2 + (x(1) - 1)^2`

The optimizer stops when it is 60 second since last time the minimum where updated.

Output:

- `[x,fval,exitflag,hessian]` : See the `nb_abc.call` method for more on these outputs.

See also:

[nb_abc.call](#)

■ **nb_bee**

Go to: [Properties](#) | [Methods](#)

A class representing a bee in the artificial bee colony (ABC) algorithm by Karaboga et. al (2012) or the constrained version by Stanarevic et. al (2015).

Constructor:

`obj = nb_bee(n);`

Input:

- `n` : Number of objects to initialize, as a scalar integer.

Output:

- `obj` : An object of class `nb_bee`.

See also:

[nb_abc](#), [nb_abc.call](#)

Properties:

- **current**
- **currentFeasible**
- **currentValue**
- **currentViolation**
- **fitness**
- **index**
- **tested**
- **testedFeasible**
- **testedValue**
- **testedViolation**
- **trials**
- **type**

- **current** ↑

Current location of the bee. Used by employed bees only.

- **currentFeasible** ↑

Indicate if the current point is feasible or not given the constraints. Default is true.

- **currentValue** ↑

Current objective value of the bee. Used by employed bees only.

- **currentViolation** ↑

Current constraint violation of the bee. Used by employed bees only.

- **fitness** ↑

The fitness at the current point. Used by employed bees only.

- **index** ↑

The index of the employed bee that the onlooker bee is dancing with. Used by onlooker bees only.

- **tested** ↑

Tested location of the bee.

- **testedFeasible** ↑

Indicate if the tested point is feasible or not given the constraints. Default is false.

- **testedValue** ↑

Objective value of the bee at the tested location.

- **testedViolation** ↑

Constraint violation of the bee at the tested location.

- **trials** ↑

The number of trials at the current area. Used by employed bees only.

- **type** ↑

The type of bee. Either 'employed' or 'onlooker'.

Methods:

- calcFitness
- drawCandidates
- initialize
- scout
- updateLocation
- calculateProbabilities
- evaluate
- joinDance
- separate
- dance
- evaluateParallel
- relocate
- tournamentSelection

► **calcFitness** ↑

```
obj = calcFitness(obj, fitnessFunc, fitnessScale, fMin)
```

Description:

Calculate fitness score at the current best value of each employed bee.

Input:

- obj : A vector of nb_bee objects.
- fitnessFunc : See the property with the same name in the nb_abc class.
- fitnessScale : See the property with the same name in the nb_abc class.
- fMin : The smallest function evaluation found up until now.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_abc.doMinimization](#)

Written by Kenneth Sæterhagen Paulsen

► **calculateProbabilities** ↑

```
prob = calculateProbabilities(employed,hasConstraints)
```

Description:

Without constraints:

Calculate the probability values for the solutions using fitness of the solutions by

```
p(i) = f(i)/sum_i(f(i))
```

See "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications", Karaboga et. al (2012).

With constraints:

Calculate the probability values for the solutions using fitness of the solutions and the constraint violations (CV) by

```
p(i) = 0.5 + f(i)/sum_i(f(i))*0.5, if solution is feasible.  
p(i) = (1 - CV(i)/sum_i(CV(i)))*0.5, if solution is infeasible.
```

See "Modified artificial bee colony algorithm for constrained problems optimization", Stanarevic et. al (2015)

Input:

- employed : A vector of employed nb_bee objects.
- hasConstraints : True if we are doing constrained maximization, or false otherwise.

Output:

- prob : The probabilities of onlooker bees to join the dance of employed bees.

See also:

[nb_bee.joinDance](#)

Written by Kenneth Sæterhagen Paulsen

► **dance ↑**

```
obj = dance(obj,lowerBound,upperBound)
```

Description:

Make the employed bees move at random in their dancing area.

See https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm for how the employed bees are dancing.

Input:

- obj : A vector of nb_bee objects.
- lowerBound : See the property with the same name in the nb_abc class.
- upperBound : See the property with the same name in the nb_abc class.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_bee.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **drawCandidates** ↑

```
draws = nb_bee.drawCandidates(lb,ub,nPar,nBees)
```

Description:

Draw candidates from a N dimensional box.

Input:

- lb : A double vector with size nPar x 1.
- ub : A double vector with size nPar x 1.
- nPar : The number of parameters.
- nBees : The number of bees.

Output:

- draws : A double with size nPar x nBees with the randomly selected parameters inside the box.

See also:

[nb_bee.initialize](#), [nb_bee.scout](#)

Written by Kenneth Sæterhagen Paulsen

► **evaluate** ↑

```
obj = evaluate(obj,objective,constrFunc)
```

Description:

Evaluate the bees at their tested locations.

Input:

- obj : A vector of nb_bee objects.
- objective : See the property with the same name in the nb_abc class.
- constrFunc : See the output from nb_abc.getConstraints.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_abc.doMinimization](#)

Written by Kenneth Sæterhagen Paulsen

► **evaluateParallel ↑**

```
obj = evaluateParallel(obj,objective,constrFunc)
```

Description:

Evaluate the bees at their tested locations.

Input:

- obj : A vector of nb_bee objects.
- objective : See the property with the same name in the nb_abc class.
- constrFunc : See the output from nb_abc.getConstraints.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_abc.doMinimization](#)

► **initialize** ↑

```
[obj,funEvals] = initialize(obj,objective,lowerBound,upperBound,...  
    objectiveLimit,maxTrials,constrFunc)
```

Description:

Initialize the bees that are going to be employed.

Input:

- obj : A vector of nb_bee objects.
- objective : See the property with the same name in the nb_abc class.
- objectiveInputs : See the property with the same name in the nb_abc class.
- lowerBound : See the property with the same name in the nb_abc class.
- upperBound : See the property with the same name in the nb_abc class.
- objectiveLimit : See the property with the same name in the nb_abc class.
- maxTrials : See the property with the same name in the nb_abc class.
- constrFunc : See the output from nb_abc.getConstraints.

Output:

- obj : A vector of nb_bee objects. The type property has been set to 'employed'.
- funEvals : Number of function evaluations during initialization.

► **joinDance** ↑

```
obj = joinDance(obj,employed,lowerBound,upperBound,hasConstraints)
```

Description:

Make the onlooker bees find a employed bee to dance with.

See https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm for how the onlooker bees are dancing.

Input:

- obj : A vector of nb_bee objects.
- lowerBound : See the property with the same name in the nb_abc class.
- upperBound : See the property with the same name in the nb_abc class.
- hasConstraints : See the property with the same name in the nb_abc class.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_bee.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **relocate ↑**

obj = relocate(obj,lowerBound,upperBound,cutLimit,hasConstraints)

Description:

Relocate the bees.

Input:

- obj : A vector of nb_bee objects.
- lowerBound : See the property with the same name in the nb_abc class.
- upperBound : See the property with the same name in the nb_abc class.
- cutLimit : See the property with the same name in the nb_abc class.
- hasConstraints : See the property with the same name in the nb_abc class.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_abc.doMinimization](#)

Written by Kenneth Sæterhagen Paulsen

► **scout** ↑

obj = scout(obj,lowerBound,upperBound,hasConstraints)

Description:

Employ scouts to new dancing areas.

See https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm for how the bees are scouting.

Input:

- obj : A vector of nb_bee objects.
- lowerBound : See the property with the same name in the nb_abc class.
- upperBound : See the property with the same name in the nb_abc class.
- hasConstraints : See the property with the same name in the nb_abc class.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_bee.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **separate** ↑

```
[employed,onlookers]      = separate(obj,cutLimit)
[employed,onlookers,scouts] = separate(obj,cutLimit)
```

Description:

Separate the bees into employed, scouts and onlookers.

Input:

- obj : A vector of nb_bee objects.
- cutLimit : See the property with the same name in the nb_abc class. Only needed if nargout > 2.

Output:

- employed : These are the employed bees that are keep dancing in their current area if nargout == 3, else it is all the employed bees.
- onlookers : These are the bees that are joing the dance of the most succesful employed bees.
- scouts : These are the employed bees that are going to scout for a new dancing area.

Written by Kenneth SÃ¶therhagen Paulsen

► tournamentSelection ↑

```
ret = tournamentSelection(obj,currentLeader)
```

Description:

Play out pairwise tournament between two bees.

We use Deb's rules to select the best candidate:

1. Any feasible solution is preferred to any infeasible solution.
2. Among two feasible solutions, the one having better objective function value is preferred.
3. Among two infeasible solutions, the one having smaller constraint violation is preferred.

Here a feasible solution does satisfies the constrains applied to the parameters, while a infeasible solution does not.

Input:

- obj : The tested bee.
- currentLeader : The bee that is currently the best.

Output:

- ret : If true is returned tested be should be made the new leader.

See also:

[nb_bee.updateLocation](#)

Written by Kenneth Sæterhagen Paulsen

► **updateLocation** ↑

obj = updateLocation(obj)

Description:

Update the location based on their last evaluation.

Input:

- obj : A vector of nb_bee objects.

Output:

- obj : A vector of nb_bee objects.

See also:

[nb_abc.doMinimization](#)

Written by Kenneth Sæterhagen Paulsen

■ **nb_optimizerDisplayer**

Go to: [Properties](#) | [Methods](#)

A class you can use to display the progress during optimization, i.e. you can pass the method handle

@(x,optVal,state) obj.update(x,optVal,state)

to the OutputFcn option to the optimizer you use. Here obj must be an object of class nb_optimizerDisplayer.

Superclasses:

handle

Constructor:

obj = nb_optimizerDisplayer(varargin)

Input:

- See the set method.

Output:

- obj : An object of class nb_optimizerDisplayer.

See also:

[fmincon](#), [fminunc](#), [fminsearch](#), [nb_fmin](#)

Written by Kenneth Sølnerhagen Paulsen

Properties:

- `axesHandle`
- `figureHandle`
- `formatDone`
- `formatIter`
- `fval`
- `includeStop`
- `includeTime`
- `iteration`
- `names`
- `notifyStep`
- `plotHandle`
- `problemSize`
- `stepLength`
- `stopped`
- `storeMax`
- `textBoxHandle`
- `type`
- `usedNorm`
- `x`

- **axesHandle** [↑](#)

The handle to the axes object where the iterations values are plotted.

- **figureHandle** [↑](#)

The handle to the figure where the optimization iterations are displayed.

- **formatDone** [↑](#)

Format of printed output when the optimizer has converged. The first input is the function value and the second input is step length measured as the norm of the change in the x-values. Then the x-values will be given as the next inputs, i.e. each element of x will be pasted as a separate input in the order they are in the vector x. At last each element of the names property will be pasted as a separate input.

E.g:

```
'Optimal value found: f(x): %1$20.6f    x: %3$20.6f\n' (Scalar)
'Optimal value found: %5$s: %3$20.6f %6$s: %4$20.6f\n' (Two)
'Optimal value found: f(x): %1$20.6f\n' (Multivariate)
```

If empty default print out is used.

Caution: Give a multi-lined char to print it one more lines.

- **formatIter** [↑](#)

Format of printed output for each stored iteration of the optimization. The first input is the iteration count, the second input is the function value and the third input is step length measured as the norm of the change in the x-values. Then the x-values will be given as the next inputs, i.e. each element of x will be pasted as a separate input in the order they are in the vector x. At last each element of the names property will be pasted as a separate input.

E.g:

```
'Iter: %1$13d  f(x): %2$20.6f    x: %4$20.6f\n' (Scalar)
```

```
'Iter: %1$13d f(x): %6$s: %4$20.6f %7$s: %5$20.6f\n' (Two)
```

```
'Iter: %1$13d f(x): %2$20.6f\n' (Multivariate)
```

If empty default print out is used.

Caution: Give a multi-lined char to print it one more lines.

- **fval** ↑

Storing the function values at each iteration. See the iteration property for which iteration that has been stored.

- **includeStop** ↑

Set to true to include a stop button in the figure when the type property is set to 'text'.

- **includeTime** ↑

Set to true to include a time left in the figure when the type property is set to 'text'.

- **iteration** ↑

Storing the iteration count of the stored values of x and fval properties.

- **names** ↑

Give the names of each element in x. Must be set to a cellstr. If generic names will be assign, i.e. {'x1','x2',...}.

See also formatDone or formatIter on how to use the names in print out.

Caution: Must either be empty or has the same length as the number of values in x.

- **notifyStep** ↑

This property sets the number of steps between the notification of the optimizer current value.

- **plotHandle** ↑

The handle to the line object with the plotted iterations.

- **problemSize** ↑

The size of the problem to solve.

- **stepLength** ↑

Stores the step length between each stored iteration. See the usedNorm property for more on how it is measured.

- **stopped** ↑

Will be set to true if the stop button of the display is pushed.

- **storeMax** ↑

Maximum number of iterations to store. Default is to store all. Minimum is 2.

- **textBoxHandle** ↑

The handle to the uicontrol object where the optimization iterations are displayed.

- **type** ↑

Set the type of reporting;
> 'text' : The reporting displayed as text in a window.
> 'command' : The reporting displayed as text in the command window
> 'graph' : The reporting displayed as a graph in a window.
> 'textgraph' : The reporting displayed as text and a graph in a window.

- **usedNorm** ↑

The used norm for calculating the step length. Default is the 2-norm.

- **x** ↑

Storing the x-values at each iteration. See the iteration property for which iteration that has been stored.

Methods:

- [getOutputFunction](#)
- [notifyError](#)
- [set](#)
- [update](#)

► **getOutputFunction** ↑

`fh = getOutputFunction(obj)`

Description:

Get function handle that can be assign to the field OutputFcn of the options struct given to the given optimizer (fminsearch, fmincon, fminunc) or solver (fsolve).

Input:

- obj : An object of class nb_optimizerDisplayer.

Output:

- fh : A function_handle object.

See also:

optimset, fminsearch, fmincon, fminunc, fsolve, nb_solver

Written by Kenneth Sæterhagen Paulsen

► **notifyError** ↑

notifyError(obj,err)

Description:

Call to notify about errors occurring optimization or solving.

Input:

- obj : An object of class nb_optimizerDisplayer
- err : The error message to display.

Written by Kenneth Sæterhagen Paulsen

► **set** ↑

set(obj,varargin)

Description:

Set properties of the object using 'propertyName', PropertyValue pairs.

Written by Kenneth Sæterhagen Paulsen

► **update** ↑

stop = update(obj,x,optVal,state,varargin)

Description:

The method you need to pass to the OutputFcn as a function handle on the form `@(x,optVal,state) obj.update(x,optVal,state)`.

Input:

- `obj` : An object of class nb_optimizerDisplayer
- `x` : The point computed by the algorithm at the current iteration
- `optVal` : Structure containing data from the current iteration.
- `state` : The current state of the algorithm ('init', 'interrupt', 'iter', or 'done')

Output:

- `stop` : Always return false.

Written by Kenneth Sæterhagen Paulsen

◆ nb_callOptimizer

```
[estPar,fval,Hessian] = ...
    nb_callOptimizer(optimizer,fh,init,lb,ub,opt,message,varargin)
[estPar,fval,Hessian,err] = ...
    nb_callOptimizer(optimizer,fh,init,lb,ub,opt,message,varargin)
```

Description:

Generic function to call different optimizers on a specific job.

Input:

- `optimizer` : A one line char with the optimizer to use. Go to see also section for a list of the supported optimizers.
- `fh` : A function handle with the function to minimize.
- `init` : Initial values on the parameters. As a nPar x 1 double.
- `lb` : Lower bound on the parameters. These may need to finite for some optimizers! As a nPar x 1 double.
- `ub` : Upper bound on the parameters. These may need to finite for some optimizers! As a nPar x 1 double.
- `opt` : A struct with the optimization options. See `nb_getDefaultOptimset`.
- `message` : Extra message given when some error happened during estimation. May be set to ''.

Optional input:

- 'Aeq' : Apply linear equality constraints. For more see the documentation of the Aeq input to the fmincon function. Only supported when optimizer is set to 'fmincon'.
- 'Beq' : Apply linear equality constraints. For more see the documentation of the Beq input to the fmincon function. Only supported when optimizer is set to 'fmincon'.
- 'NONLCON' : Apply non-linear constraints. For more see the documentation of the constraints property of the nb_abc class or the NONLCON input to the fmincon function. Only supported when optimizer is set to 'nb_abc' or 'fmincon'.
- varargin : Optional inputs given to the function handle fh when being evaluated.

Output:

- estPar : The values of the parameters that minimizes the function. As a nPar x 1 double.
- fval : Value of the function at the minimum. As a scalar double.
- Hessian : Estimate of the Hessian at the found minimum.
- err : Provide this output to return error instead of throwing it inside this function.

See also:

[nb_abc](#), [nb_fmin](#), [nb_pso](#), [fmincon](#), [fminunc](#), [fminsearch](#), [bee_gate](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_getDefaultOptimset**

```
opt = nb_getDefaultOptimset(funcName)
opt = nb_getDefaultOptimset(opt,funcName)
```

Description:

If the opt input is empty or a struct without any fields, then this method will return the default setting for the optimizer/solver used.

I.e. for the MATLAB functions 'fmincon', 'fminunc', 'fminsearch' and 'fsolve' this amounts to calling optimset with the name of the function as the only input.

If this function doesn't recognize the function given by funcName, it will just return the opt input as it is (except that it will add the 'Display' and 'OutputFcn' fields if not already added).

Input:

- opt : A struct (with or without fields) or any other empty object.
- funcName : Name of the optimizer/solver to be used.

Output:

- opt : A struct with the options used by the optimizer/solver.

See also:

[optimset](#), [nb_abc.optimset](#), [nb_solve.optimset](#), [nb_abcSolve.optimset](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_getOptimizers**

`optimizers = nb_getOptimizers(type)`

Description:

Get the optimizers you have available for different estimation jobs.

This function also tries to detect optimizer of other toolboxes, such as csminwel (Chris Sims) and bee_gate (RISE toolbox). You need to add these optimizers separately, as they are not part of NB toolbox.

Input:

- type : Either 'arima', 'ml' or '' (all). When set to 'arima' or 'ml' optimizers that needs fixed bound, as nb_abc, are not returned.

Output:

- optimizers : A cellstr with the supported optimizers.

See also:

[nb_getDefaultOptimset](#), [nb_abc](#), [nb_pso](#), [fmincon](#), [fminunc](#), [fminsearch](#)

Written by Kenneth Sætherhagen Paulsen

◆ **nb_interpretExitFlag**

```
nb_interpretExitFlag(e,type)
message = nb_interpretExitFlag(e,type,extra,homotopyErr)
```

Description:

Interpret error from given optimizers or zero root finders.

If nargout == 1, the error will not be thrown! Instead the error message will be returned.

Input:

- e : The exit flag of the given optimizer or solver.
- type : The optimizer or solver used as a string. E.g. 'fmincon', 'fminsearch', 'fsolve', etc.
- extra : A string with extra information on the error. Will be appended. Default is ''.
- homotopyErr : If nb_homotopy is used you can provide the err output of that function as this input.

Output:

- message : If asked for the error message is returned instead of thrown.

Written by Kenneth Sæterhagen Paulsen

◆ nb_optimizerMessageState

nb_optimizerMessageState(value)

Description:

If state is set to 'true'/true, the nb_interpretExitFlag and nb_callOptimizer functions only throws warnings instead of errors

Caution: Ignored if the error message is asked to be returned as a one line char!.

Input:

- e : The exit flag of the given optimizer or solver.
- type : The optimizer or solver used as a string. E.g. 'fmincon', 'fminsearch', 'fsolve', etc.
- extra : A string with extra information on the error. Will be appended. Default is ''.
- homotopyErr : If nb_homotopy is used you can provide the err output of that function as this input.

Written by Kenneth Sæterhagen Paulsen

◆ nb_pso

```
[x,fval,exitflag] = nb_pso(fh,init,lb,ub,options,varargin)
```

Description:

Find the minimum of a function using Particle Swarm Optimization.

This is a wrapper function of the pso.do function made by S. Samuel Chen.

This makes it possible to add optional input arguments to the optimizer function, which is not possible to do in pso.do.

Input:

- fh : The objective to minimize. As a function handle.
- init : The initial values of the optimization. Default is zero. As a n x 1 double.
- lb : The lower bound on the x values. Set it to -inf for the elements in x that is not constrained. Default is to set it to -inf for all elements. As a n x 1 double.
- ub : The upper bound on the x values. Set it to inf for the elements in x that is not constrained. Default is to set it to inf for all elements. As a n x 1 double.
- options : See the pso.optimset method for more on this input.

Optional inputs:

- varargin : Optional number of inputs given to the objective when it is called during the minimization algorithm.

Output:

- x : The point at the found minimum, as a n x 1 double.
- fval : The value of the objective at the found minimum, as a scalar double.
- exitflag : See the documentation of the exitFlag property of the pso.do class.

Examples:

```
g = @(x)-sin(x);
x1 = nb_pso(g,1)
x2 = nb_pso(g,-3)
x3 = nb_pso(g,-3,-4,4)

opt.Display = 'final';
x4 = nb_pso(g,-3,-4,4,opt)
```

See also:

[pso.do](#), [pso.optimset](#), [fmincon](#), [fminunc](#), [fminsearch](#), [bee_gate](#)

Written by Kenneth Sætherhagen Paulsen

25.20 Parallel coding

- WorkerObjWrapper
 - nb_closePool
 - nb_funcToWrite
 - nb_parCheck
 - nb_updateWaitbarParallel
 - nb_availablePoolSize
 - nb_fileToWrite
 - nb_openPool
 - nb_poolSize
-

◆ nb_availablePoolSize

```
s = nb_availablePoolSize()
```

Description:

Get number of available workers when running in parallel.

Output:

- s : An interger with the number of workers.

Caution: If the environment variable clusterProfile is set, then inf is returned, as in this case the number of cores used cannot be set anyway.

See also:

[nb_openPool](#), [nb_closePool](#), [nb_poolSize](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_closePool

```
nb_closePool(ret)
```

Description:

Close a parallel session. Invariante to MATLAB version.

Input:

- ret : 1 if the matlabpool should be close otherwise 0. Default is 1.

See also:

[nb_openPool](#), [nb_poolSize](#), [nb_availablePoolSize](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_fileToWrite

```
h = nb_fileToWrite(name)
```

Description:

Get handle to a file to communicate with when running in parallel

Input:

- name : Name of file to write to. As a string.
- gui : Give 'gui' to place the file in the nb_userpath('gui') folder.

Output:

- h : A file identifier object. See fopen.

Written by Kenneth Sæterhagen Paulsen

◆ nb_funcToWrite

```
w = nb_funcToWrite(name,gui)
```

Description:

Get handle to a function that can communicate with a file when running in parallel. See

Input:

- name : Name of file to write to. As a string.
- gui : Give 'gui' to place the file in the nb_userpath('gui') folder.

Output:

- w : A WorkerObjWrapper object.

Example:

```
w = nb_funcToWrite('test','gui');
parfor ii = 1:10
    fprintf(w.Value,'Some text');
end
clear w;
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_openPool

```
ret = nb_openPool()      : Uses all cores
ret = nb_openPool(cores) : Uses the number of selected cores
```

Description:

Open up a parallel session if not already open.

Caution: To open a cluster pool set the environment variable
clusterProfile; setenv('clusterProfile','yourProfileName')

Input:

- cores : Number of cores to open, as an integer. Default is available.

Output:

- ret : 1 if matlabpool was opened, else 0.

See also:

[nb_closePool](#), [nb_poolSize](#), [nb_availablePoolSize](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_parCheck**

```
[ret,D] = nb_parCheck()
```

Description:

Test if it is possible to make a waitbar in parfor using the
parallel.pool.DataQueue class or not.

Output:

- ret : true or false

- D : Handle to the parallel.pool.DataQueue object if ret is true,
otherwise D is empty.

See also:

[nb_waitbar](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_poolSize**

```
s = nb_poolSize()
```

Description:

Get number of active workers when running in parallel.

Output:

- s : An interger with the number of workers.

See also:

[nb_openPool](#), [nb_closePool](#), [nb_availablePoolSize](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_updateWaitbarParallel**

`nb_updateWaitbarParallel(note,h)`

Description:

Function that can be given to the `afterEach` method of the `parallel.pool.DataQueue` to update the `nb_waitbar` class during parfor.

Input:

- note : The increment at update.
- h : The handle to the `nb_waitbar` object.

Examples:

```
D = parallel.pool.DataQueue;
afterEach(D,@(x)nb_updateWaitbarParallel(x,h));
```

See also:

[nb_waitbar](#)

Written by Kenneth Sæterhagen Paulsen

25.21 Solvers

- nb_abcSolve
- nb_blockDecompose
- nb_homotopy
- nb_lyapunovEquation2
- nb_perfectForesight.FDeriv
- nb_perfectForesight.FEndDeriv
- nb_perfectForesight.FEpiDeriv
- nb_perfectForesight.FEqsDeriv
- nb_perfectForesight.FFirstDeriv
- nb_perfectForesight.blockSolution
- nb_perfectForesight.doNewtonUpdate
- nb_perfectForesight.eqs2funcEpi
- nb_perfectForesight.getEqOfPeriod
- nb_perfectForesight.getFunctionValue
- nb_perfectForesight.getJacobianOnePeriod
- nb_perfectForesight.getNewtonUpdateOnePeriod
- nb_perfectForesight.getSolvedPrologue
- nb_perfectForesight.getStackedFirst
- nb_perfectForesight.getStackedJacobianAutomatic
- nb_perfectForesight.getStackedSystemFunction
- nb_perfectForesight.normalIteration
- nb_perfectForesight.normalSolutionStochInitVal
- nb_perfectForesight.proSolver
- nb_perfectForesight.unexpectedSolver
- nb_perfectForesight.updateStackedSystemFunction
- nb_perfectForesight.updateSystemFunctionOnePeriod
- nb_solveLinearRationalExpModel
- nb_beeSolver
- nb_getSolvers
- nb_lyapunovEquation
- nb_perfectForesight.F
- nb_perfectForesight.FEnd
- nb_perfectForesight.FEpi
- nb_perfectForesight.FEqs
- nb_perfectForesight.FFirst
- nb_perfectForesight.blockIteration
- nb_perfectForesight.checkJacobian
- nb_perfectForesight.epiSolver
- nb_perfectForesight.forwardSolution
- nb_perfectForesight.getExo
- nb_perfectForesight.getFunctionValueOnePeriod
- nb_perfectForesight.getNewtonUpdate
- nb_perfectForesight.getParameters
- nb_perfectForesight.getStackedEnd
- nb_perfectForesight.getStackedJacobian
- nb_perfectForesight.getStackedJacobianSymbol
- nb_perfectForesight.getStartingValues
- nb_perfectForesight.normalSolution
- nb_perfectForesight.optimset
- nb_perfectForesight.substituteSS
- nb_perfectForesight.updateEpiFunctionOnePeriod
- nb_perfectForesight.updateStackedSystemFunction
- nb_solve
- nb_solver

► nb_perfectForesight.F ↑

```
Y = nb_perfectForesight.F(G, vars, parVal, exoVal, ...
                           initVal, varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FDeriv** ↑

Y = nb_perfectForesight.FDeriv(GDeriv,vars,parVal,exoVal,initVal,varsSS)

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEnd** ↑

Y = nb_perfectForesight.FEnd(GEnd,vars,parVal,exoVal,endVal,varsss)

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEndDeriv** ↑

Y = nb_perfectForesight.FEndDeriv(GEnd,vars,parVal,exoVal,endVal,varsSS)

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEpi** ↑

Y = nb_perfectForesight.FEpi(G,epivars,vars,pars,ssVars)

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEpiDeriv ↑**

```
Y = nb_perfectForesight.FDeriv(GDeriv,epivars,vars,pars,ssVars)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEqs ↑**

```
Y = nb_perfectForesight.FEqs(GEqs,vars,parVal,exoVal,varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FEqsDeriv ↑**

```
Y = nb_perfectForesight.FEqsDeriv(GEqsDeriv,vars,parVal,exoVal,varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.FFirst ↑**

```
Y = nb_perfectForesight.FFirst(GFirst,vars,parVal,exoVal,...  
initVal,varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.FFirstDeriv** ↑

```
Y = nb_perfectForesight.FFirstDeriv(GFirstDeriv,vars,parVal,exoVal,...  
initVal,varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.blockIteration** ↑

```
[YT,err] = nb_perfectForesight.blockIteration(obj,funcs,inputs,iter,...  
outer,YT)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.blockSolution** ↑

```
Y = nb_perfectForesight.blockSolution(obj,Y,inputs)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.checkJacobian** ↑

```
nb_perfectForesight.checkJacobian(obj,JFM)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.doNewtonUpdate ↑**

```
[DY,err] = nb_perfectForesight.doNewtonUpdate(FY,JF,solveIter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.epiSolver ↑**

```
[Y,err] = nb_perfectForesight.epiSolver(obj,inputs,Y,iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.eqs2funcEpi ↑**

```
[eqs,funcs] = nb_perfectForesight.eqs2funcEpi(inputs,allEndo,epiVars,...  
vars,varsExo,pars,eqs)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.forwardSolution ↑**

```
[Y,err] = nb_perfectForesight.forwardSolution(obj,Y,inputs,eqs,vars,...  
exoVal,iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getEqOfPeriod** ↑

```
[out,varsN] = nb_perfectForesight.getEqOfPeriod(eqs,matches,pars,...  
paramN,varsExo,varsExoN,vars,varsN)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getExo** ↑

```
[varsExo,varsExoN,varsExoNS] = nb_perfectForesight.getExo(parser)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getFunctionValue** ↑

```
FY = nb_perfectForesight.getFunctionValue(Y,funcs,inputs,iter)
```

Description:

Get function evaluation of stacked system of equations.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getFunctionValueOnePeriod** ↑

```
FY = nb_perfectForesight.getFunctionValueOnePeriod(Y,exoVal,funcs)
```

Description:

Get function evaluation of one period of the system of equations.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getJacobianOnePeriod** ↑

```
JF = nb_perfectForesight.getJacobianOnePeriod(funcs,Y,varargin)
```

Description:

Evaluate the Jacobian at the current iteration

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getNewtonUpdate** ↑

```
[DY,err] = nb_perfectForesight.getNewtonUpdate(funcs,JF,Y,inputs,...  
solveIter,iter)
```

Description:

Get the Newton update utilizing sparsity of the jacobian JF.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getNewtonUpdateOnePeriod** ↑

```
[DY,err] = nb_perfectForesight.getNewtonUpdateOnePeriod(funcs,JF,...  
solveIter,Y,varargin)
```

Description:

Get the Newton update for one period utilizing sparsity of the jacobian JF

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **[nb_perfectForesight.getParameters](#)** ↑

```
[paramN,paramNS,pars] = nb_perfectForesight.getParameters(parser)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **[nb_perfectForesight.getSolvedPrologue](#)** ↑

```
inputs = nb_perfectForesight.getSolvedPrologue(obj,inputs,YT,proY,iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **[nb_perfectForesight.getStackedEnd](#)** ↑

```
[stackedEnd,endN,varsN] = nb_perfectForesight.getStackedEnd(eqs,...  
matches,pars,paramN,varsExo,varsExoN,varsCLag,endVal)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **[nb_perfectForesight.getStackedFirst](#)** ↑

```
[stackedFirst,initN,varsN] = nb_perfectForesight.getStackedFirst(eqs,...  
matches,pars,paramN,varsExo,varsExoN,varsCLead,initVal)
```

Syntax:

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.getStackedJacobian** ↑

```
JF = nb_perfectForesight.getStackedJacobian(Y,inputs,funcs,iter)
```

Description:

Evaluate the Jacobian at the current iteration.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_perfectForesight.getStackedJacobianAutomatic** ↑

```
[YT,err] = nb_perfectForesight.blockIteration(obj,funcs,inputs,iter,...  
      outer,YT)
```

Description:

Evaluate the Jacobian at the current iteration using automatic derivatives that handles sparsity. For more see the myAD class written by SeHyoun Ahn and Martin Fink.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_perfectForesight.getStackedJacobianSymbolic** ↑

```
JF = nb_perfectForesight.getStackedJacobianSymbolic(inputs,funcs,Y,iter)
```

Description:

Evaluate the Jacobian at the current iteration using symbolic derivatives that handles sparsity. For more see the nb_mySD and nb_Param classes written by Kenneth S. Paulsen.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶therhagen Paulsen

► **nb_perfectForesight.getStackedSystemFunction** ↑

```
[funcs,inputs] = nb_perfectForesight.getStackedSystemFunction(obj,...  
      inputs,parser)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.getStartingValues** ↑

```
Y = nb_perfectForesight.getStartingValues(obj,inputs)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.normalIteration** ↑

```
[YT,err] = nb_perfectForesight.normalIteration(obj,funcs,inputs,iter,...  
outer,YT)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.normalSolution** ↑

```
[YT,err] = nb_perfectForesight.normalSolution(obj,Y,inputs,funcs)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.normalSolutionStochInitVal** ↑

```
[YT,err] = nb_perfectForesight.normalSolutionStochInitVal(obj,Y,...  
inputs,funcs)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.optimset** ↑

```
opt = nb_getDefaultOptimset(solver)
```

Description:

Get default options for solving the perfect foresight problem.

Input:

- solver : Name of the solver to be used. Use nb_solve.optimset('help') or nb_abcSolve.optimset('help') for more on the different options.

Output:

- opt : A struct with the options used by the solver.

See also:

[nb_solve.optimset](#), [nb_abcSolve.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.proSolver** ↑

```
[Y,err] = nb_perfectForesight.proSolver(obj,inputs,Y,iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.substituteSS** ↑

```
[YT,err] = nb_perfectForesight.blockIteration(obj,funcs,inputs,iter,...  
outer,YT)
```

Description:

Here we need to substitute in for the end values where the steady-state operator is used.

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► nb_perfectForesight.unexpectedSolver ↑

```
[YT,err] = nb_perfectForesight.unexpectedSolver(obj,funcs,inputs,Y,...  
funcHandle)
```

Description:

Solver used when we are dealing with unexpected shocks.

Part of the perfect forseight solver package nb_perfectForesight.

Input:

- funcHandle : Either @nb_perfectForesight.normalIteration or
@nb_perfectForesight.blockIteration

Written by Kenneth Sæterhagen Paulsen

► nb_perfectForesight.updateEpiFunctionOnePeriod ↑

```
funcs = nb_perfectForesight.updateEpiFunctionOnePeriod(funcs,...  
Yepi,YNotEpi,parExoVal,varsSS)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► nb_perfectForesight.updateStackedSystemFunction ↑

```
funcs = nb_perfectForesight.updateStackedSystemFunction(obj,funcs,...  
inputs,iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth Sæterhagen Paulsen

► **nb_perfectForesight.updateStackedSystemFunctionOnePeriod** ↑

```
funcs = nb_perfectForesight.updateStackedSystemFunctionOnePeriod(obj, ...
    funcs, inputs, init, iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

► **nb_perfectForesight.updateSystemFunctionOnePeriod** ↑

```
funcs = nb_perfectForesight.updateSystemFunctionOnePeriod(obj, ...
    funcs, inputs, init, exoVal, iter)
```

Description:

Part of the perfect forseight solver package nb_perfectForesight.

Written by Kenneth SÃ¶terhagen Paulsen

■ **nb_abcSolve**

Go to: [Properties](#) | [Methods](#)

An implementation of the artificial bee colony algorithm for solving a multivariate non-linear system of equations.

References:

"A comprehensive survey: Artificial bee colony (ABC) algorithm and applications", Karaboga et. al (2012).

"Modified artificial bee colony algorithm for constrained problems optimization", Stanarevic et. al (2015)

This class tries to solve the following problem

```
F(x) = 0
x in S
```

where $F(x)$ is a set of non-linear solutions. S is the search space, i.e. the space to look for candidate values to solve $F(x) = 0$.

Caution: The steps of this implementation is slightly different from that of Karaboga et. al (2012).

Initialize employed bees by scouting.

Repeat:

1. Employed bees that has reached the cutLimit are sent out scouting. See nb_beeSolver.scout (local == false) or nb_beeSolve.joinDance (local == true).
2. Send onlookers to join the employed bees by random. Higher probabilities are assign to the employed bees that communicates good values of the objective. Finally onlookers are sent to explore the neighbourhood of the selected employed bee See nb_beeSolver.joinDance.
3. Employed bees are then sent to explore their neighbourhood by changing the current value of x. This is done by changing a random number of elements in x, the move is selected using $x(n) = x(n-1) + \alpha * F(x)$, where alpha is random search direction matrix. Each element is drawn from the uniform distribution in [-1,1]. See nb_beeSolver.dance.
4. All the new points of the bees are evaluated. (This is the step that can be ran in parallel.)
5. The solution is then selected and stored.

This is repeated until some of the limits (maxTime, maxFunEvals, maxIter, maxSolutions or maxTimeSinceUpdate are reached).

Superclasses:

handle

Constructor:

```
obj = nb_abcSolve(fh,init,lb,ub)
obj = nb_abcSolve(fh,init,lb,ub,options,varargin)
obj = nb_abcSolve(fh,init,[],[],options,varargin)
```

Input:

- fh : See the documentation of the objective property of the nb_abcSolve class.
- init : See the documentation of the initialXValue property of the nb_abcSolve class.
- lb : See the documentation of the lowerBound property of the nb_abcSolve class.
- ub : See the documentation of the upperBound property of the nb_abcSolve class.
- options : See the nb_abcSolve.optimset method for more on this input.

Optional inputs:

- varargin : Optional number of inputs given to the objective when it is called during the minimization algorithm.

Output:

- obj : An object of class nb_abcSolve.

See also:

[nb_abcSolve.optimset](#), [nb_abcSolve.call](#), [nb_solve](#), [fsolve](#)

Written by Kenneth Sølnerhagen Paulsen

Properties:

- **F**
- **bees**
- **cutLimit**
- **display**
- **display**
- **employedShare**
- **exitFlag**
- **fitnessFunc**
- **fitnessScale**
- **funEvals**
- **initialXValue**
- **iterations**
- **jacobianFunction**
- **local**
- **lowerBound**
- **maxFunEvals**
- **maxIter**
- **maxNodes**
- **maxSolutions**
- **maxTime**
- **maxTimeSinceUpdate**
- **maxTrials**
- **meritFunction**
- **meritFunctionValue**
- **minFFunctionValue**
- **minFunctionValue**
- **minSolutions**
- **minXValue**
- **newtonShare**
- **newtonStop**
- **numSolutions**
- **numWorkers**
- **objectiveLimit**
- **options**
- **tolerance**
- **toleranceX**
- **upperBound**
- **useParallel**
- **xValue**

- **F** ↑

The function to solve. As a function handle. Must take a N x 1 double as the first input, and return a N x 1 double.

- **bees** ↑

A vector of `nb_beeSolver` objects.

- **cutLimit** ↑

Positive integer. The maximum trial that is allowed for employed bee before it starts to scout for another area. Default is 20. Must be greater than 4.

- **display** ↑

'iter' Level of display. 'iter' displays output at each iteration. Only supported option.

- **display** ↑

An object of class `nb_optimizerDisplayer`. This object set the way the optimizer displays result during the optimization.

- **employedShare** ↑

This set the share of employed bees. Default is 0.33, i.e. 33%. Must be in the set (0.1,0.9).

- **exitFlag** ↑

The reason for exiting. One of:

```
> 6 : Maximum number of solutions exceeded.  
> 5 : User stop the optimization manually with finding of at  
      least the minimum number of solution asked for.  
> 4 : Maximum time limit since last update exceeded  
      (maxTimeSinceUpdate) with finding of at least the  
      minimum number of solution asked for.  
> 3 : Maximum time limit exceeded (maxTime) with finding of at  
      least the minimum number of solution asked for.  
> 2 : Maximum iteration limit exceeded (maxIter) with finding  
      of at least the minimum number of solution asked for.  
> 1 : Maximum function evaluation limit exceeded (maxFunEvals)  
      with finding of at least the minimum number of solution  
      asked for.  
> -1 : Maximum function evaluation limit exceeded (maxFunEvals)  
      without finding the minimum number of solution  
      asked for.  
> -2 : Maximum iteration limit exceeded (maxIter) without  
      finding the minimum number of solution asked for.  
> -3 : Maximum time limit exceeded (maxTime) without finding  
      the minimum number of solution asked for.  
> -4 : Maximum time limit since last update exceeded  
      (maxTimeSinceUpdate) without finding the minimum number  
      of solution asked for.  
> -5 : User stop the optimization manually without finding  
      the minimum number of solution asked for.
```

- **fitnessFunc** ↑

A function handle that return the fitness of each bee at each iteration of the optimization. See for example `nb_fitness.standard`, which is the default fitness function. Other proper fitness functions may be found in the `nb_fitness` package.

- **fitnessScale** ↑

This set the value of the fitness scaling parameter. Default is 1. This is the third input to the function given by the `fitnessFunc` property.

- **funEvals** ↑

Number of function evaluations.

- **initialXValue** ↑

The initial values of the optimization. Default is zero. As a N x 1 double.

- **iterations** ↑

Number of iteration of the abc algorithm.

- **jacobianFunction** ↑

A function handle that return the jacobian at the current evaluation point. Must return a N x N double or sparse matrix. Default is [], i.e. the newton type bees used a Steffensen update instead.

- **local** ↑

Use the algorithm for finding a local solutions, i.e. if set to true, this will not search globally in S, but instead search for a solution in the neighborhood of the current best candidate of the employed bees. During initialization the bees are not sent out to search the full space S at random, but instead doing random small steps away from the initial value.

- **lowerBound** ↑

The lower bound on the x values. As a N x 1 double. Must be finite!

- **maxFunEvals** ↑

Positive integer. Maximum number of function evaluation allowed. Default is inf.

- **maxIter** ↑

Positive integer. Maximum number of iterations allowed. Default is inf.

- **maxNodes** ↑

Positive integer. The number of bees to use during optimization.

- **maxSolutions** ↑

Positive integer. The maximum number of solutions to allow. To trigger a local solution algorithm set this to 1 and local to true. Default is 20.

- **maxTime** ↑

Positive integer. The maximum time spent on optimization in seconds. Default is 60 seconds.

- **maxTimeSinceUpdate** ↑

Positive integer. The maximum time spent on optimization since last time the minimum point where updated (in seconds).

- **maxTrials** ↑

Positive integer. The maximum trial that is allowed for finding initial candidates. Default is 100.

- **meritFunction** ↑

A function handle with the merit function to be applied to test for solution candidates. A G : R^N → R function. Default is @(x)norm(x,2).

- **meritFunctionValue** ↑

The merit function value of the solutions found.

- **minFFunctionValue** ↑

The value of F(minXValue). As a N x 1 double.

- **minFunctionValue** ↑

The minimized merit function value found during the solving. As a 1 x nSolutions double.

- **minSolutions** ↑

Positive integer. The minimum number of solutions to find. An negative exitFlag will be thrown if the number of solutions are not greater then or equal to this number. Default is 1.

- **minXValue** ↑

The x values at the located minimum of the merit function. As a N x 1 double.

- **newtonShare** ↑

Share of employed bees being of newton type, i.e. use a newton step of getting a new solution candidate instead of random search in the dancing area. If the derivative of the function F(x) is everywhere defined, you should set it to 1. Default is 0.1, i.e. 10%.

- **newtonStop** ↑

Set to true to make the employed bees that uses newton update to move to convert too random shearch if they breach the cutLimit without finding a solution to the problem.

- **numSolutions** ↑

Number of solutions found (N).

- **numWorkers** ↑

Positive integer. Number of worker to use when running optimization in parallel.

- **objectiveLimit** ↑

Double. If the objective return a value greater than this, a new candidate of x is drawn. Default is 1e9.

- **options** ↑

A struct with the options of the optimiziation. See the nb_abcSolve.optimset method for more on this option.

- **tolerance** ↑

Tolerance value for accepting a solution.

- **toleranceX** ↑

Tolerance value for differencing between different solutions.

- **upperBound** ↑

The upper bound on the x values. As a n x 1 double. Must be finite!

- **useParallel** ↑

true or false. Give true if the nodes should be spread on different parallel workers.

- **xValue** ↑

The suggested solution of x. As a N x nSolutions double.

Methods:

- call • optimset • set • solve • test
- test2 • test2local • test3 • testParallel

► **call** ↑

```
[x,fval,exitflag] = nb_abcSolve.call(F,init,lb,ub,options,varargin)
```

Description:

Call the artificial bee colony algorithm to solve a multivariate non-linear system of equations ($F(x) = 0$). See the documentation of the nb_abcSolve class for more on this algorithm.

Input:

- F : See the documentation of the objective property of the nb_abcSolve class.
- init : See the documentation of the initialXValue property of the nb_abcSolve class.
- lb : See the documentation of the lowerBound property of the nb_abcSolve class.
- ub : See the documentation of the upperBound property of the nb_abcSolve class.
- options : See the nb_abcSolve.optimset method for more on this input.

Optional inputs:

- varargin : Optional number of inputs given to the objective when it is called during the solving algorithm.

Output:

- x : See the documentation of the minXValue property of the nb_abcSolve class.
- fval : See the documentation of the minFunctionValue property of the nb_abcSolve class.
- exitflag : See the documentation of the exitFlag property of the nb_abcSolve class.

Examples:

```
g           = @(x)-sin(x);
opt         = nb_getDefaultOptimset('nb_abcSolve');
opt.maxSolutions = 3;
x           = nb_abcSolve.call(g,1,-4,4,opt)
opt.jacobianFunction = @(x)-cos(x);
x2          = nb_abcSolve.call(g,1,-4,4,opt)
```

See also:

[nb_abcSolve.optimset](#), [nb_solve.call](#), [fsolve](#)

Written by Kenneth Sæterhagen Paulsen

► [optimset ↑](#)

```
options = nb_abcSolve.optimset(varargin)
```

Description:

Get solving settings or help.

Optional input:

- Run without inputs to get defaults.
- Run with 'list' as the first input to get a cellstr with the supported options.
- Run with 'help' to get a struct with help on each option.
- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:

[nb_abcSolve.options](#)

Written by Kenneth Sæterhagen Paulsen

► **set ↑**

```
obj = set(obj,varargin)
```

Description:

Set properties of the object using 'propertyName', PropertyValue pairs.

Input:

- obj : An object of class nb_abcSolve.

Optional inputs:

- varargin: 'propertyName', PropertyValue pairs

Output:

- obj : An object of class nb_fmin where the wanted properties has been set.

Examples:

```
set(obj,'initialXValue',2,'lowerBound',0)
```

See also:

[nb_abcSolve.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► **solve** ↑

```
solve(obj)
solve(obj,varargin)
```

Description:

Run solving of the given problem.

Input:

- obj : An object of class nb_abcSolve.

Optional input:

- varargin : Given to the nb_abcSolve.set method.

Output:

The provided object of class nb_abcSolve has been updated the following properties;
> xValue
> meritFunctionValue
> exitFlag

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **test** ↑

```
[x,fval,exitflag] = nb_abcSolve.test()
```

Description:

Test the ABC algorithm on the Rosenbrock Vally function;

```
f = 100*(x(2) - x(1)^2)^2 + (x(1) - 1)^2
```

The solver stops when it is 60 second since last time the minimum where updated.

Output:

- [x,fval,exitflag] : See the nb_abcSolve.call method for more on these outputs.

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **test2** ↑

```
[x,fval,exitflag] = nb_abcSolve.test2(varargin)
```

Description:

Test method for solving

```
min(x(1) + x(2),0.01) = 0  
x(1).^2 - x(2) = 0
```

Uses only newton updating bees!

Output:

- [x,fval,exitflag] : See the nb_abcSolve.call method for more on these outputs.

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **test2local** ↑

```
[x,fval,exitflag] = nb_abcSolve.test2local(varargin)
```

Description:

Test method for solving

```
min(x(1) + x(2),0.01) = 0  
x(1).^2 - x(2) = 0
```

without bounds, i.e. a local method must be used.

Output:

- [x,fval,exitflag] : See the nb_abcSolve.call method for more on these outputs.

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **test3** ↑

```
[x,fval,exitflag] = nb_abcSolve.test3(varargin)
```

Description:

Test method for solving

```
max(min(x(1) + x(2),0.01),-0.01) = 0  
x(1).^2 - x(2) = 0
```

Output:

- [x,fval,exitflag] : See the nb_abcSolve.call method for more on these outputs.

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **testParallel** ↑

```
nb_abcSolve.testParallel()
```

Description:

Test the ABC algorithm in parallel on the Rosenbrock Vally function;

```
f = 100*(x(2) - x(1)^2)^2 + (x(1) - 1)^2
```

Output:

- [x,fval,exitflag,hessian] : See the nb_abcSolve.call method for more on these outputs.

See also:

[nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

■ nb_beeSolver

Go to: [Properties](#) | [Methods](#)

A class representing a bee in the artificial bee colony (ABC) algorithm by Karaboga et. al (2012) applied to solving the problem

```
F(x) = 0  
s.t. x in S
```

where $F(x)$ is a set of non-linear solutions. S is the search space, i.e. the space to look for candidate values to solve $F(x) = 0$.

Constructor:

```
obj = nb_beeSolver(n);
```

Input:

- n : Number of objects to initialize, as a scalar integer.

Output:

- obj : An object of class nb_beeSolver.

See also:

[nb_abcSolve](#), [nb_abcSolve.call](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | |
|----------------|--------------------|----------------|-----------|
| • current | • currentFValue | • currentValue | • fitness |
| • index | • maxNumberChanged | • method | • tested |
| • testedFValue | • testedValue | • trials | • type |

- **current** ↑

Current location of the bee. Used by employed bees only. A double with size N x 1.

- **currentFValue** ↑

F(currentValue), as a double with size N x 1.

- **currentValue** ↑

Current merit function value of the bee. Used by employed bees only. As a scalar double.

- **fitness** ↑

The fitness at the current point. Used by employed bees only.

- **index** ↑

The index of the employed bee that the onlooker bee is dancing with. Used by onlooker bees only.

- **maxNumberChanged** ↑

Maximum number of changed parameters at a given dance. If empty It gets the value N.

- **method** ↑

Method to use for updating the current point.

> 1 : Random search.

> 2 : Newton (either with numerical jacobian or function handle).
The function handle may return a sparse matrix.

- **tested** ↑

Tested location of the bee. A double with size N x 1.

- **testedFValue** ↑

F(testedValue), as a double with size N x 1.

- **testedValue** ↑

Objective value of the bee at the tested location.

- **trials** ↑

The number of trials at the current area. Used by employed bees only.

- **type** ↑

The type of bee. Either 'employed' or 'onlooker'.

Methods:

- calcFitness
- dealZeros
- evaluate
- joinBestDancer
- scout
- updateLocation
- calculateProbabilities
- drawCandidates
- evaluateParallel
- joinDance
- separate
- dance
- drawCandidatesLocal
- initialize
- relocate
- tournamentSelection

► **calcFitness** ↑

```
obj = calcFitness(obj, fitnessFunc, fitnessScale, fMin)
```

Description:

Calculate fitness score at the current best value of each employed bee.

Input:

- obj : A vector of nb_beeSolver objects.
- fitnessFunc : See the property with the same name in the nb_abcSolve class.
- fitnessScale : See the property with the same name in the nb_abcSolve class.
- fMin : The smallest function evaluation found up until now.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_abcSolve.doSolving](#)

Written by Kenneth Sæterhagen Paulsen

► **calculateProbabilities** ↑

```
prob = calculateProbabilities(employed)
```

Description:

Calculate the probability values for the solutions using fitness of the solutions by

```
p(i) = f(i)/sum_i(f(i))
```

where $f(i)$ is the value of the fitness of the value of the merit function by employed bee number i .

See "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications", Karaboga et. al (2012).

Input:

- employed : A vector of employed bees. As a vector of nb_beeSolver objects.

Output:

- prob : The probabilities of onlooker bees to join the dance of employed bees.

See also:

[nb_beeSolver.joinDance](#)

Written by Kenneth Sæterhagen Paulsen

► **dance ↑**

```
[employed,newScouts] = dance(employed,lowerBound,upperBound,F,...  
jacobianFunction)
```

Description:

Make the employed bees move at random in their dancing area or some move by some other updating rule.

Input:

- employed : A vector of employed bees. As a vector of nb_beeSolver objects.
- lowerBound : See the property with the same name in the nb_abcSolve class.
- upperBound : See the property with the same name in the nb_abcSolve class.
- jacobianFunction : See the property with the same name in the nb_abcSolve class.

Output:

- employed : A vector of nb_beeSolver objects.
- newScouts : A vector of nb_beeSolver objects. These are the ones that breach the bounds during the newton update.

See also:

[nb_beeSolver.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► dealZeros ↑

obj = dealZeros(obj, nPar)

Description:

Initialize the bees to the trivial solution.

Input:

- obj : A vector of nb_beeSolver objects.
- nPar : Number of values to find the solution of.

Output:

- obj : A vector of nb_beeSolver objects.

Written by Kenneth Sæterhagen Paulsen

► drawCandidates ↑

draws = nb_beeSolver.drawCandidates(lb, ub, nPar, nBees)

Description:

Draw candidates from a N dimensional box.

Input:

- lb : A double vector with size nPar x 1.
- ub : A double vector with size nPar x 1.
- nPar : The number of parameters.
- nBees : The number of bees.

Output:

- draws : A double with size nPar x nBees with the randomly selected parameters inside the box.

See also:

[nb_beeSolver.initialize](#), [nb_beeSolver.scout](#)

Written by Kenneth Sæterhagen Paulsen

► drawCandidatesLocal ↑

```
draws = nb_beeSolver.drawCandidates(fVal,cVal,x,nBees)
```

Description:

Draw candidates from a stochastic searching rule

$$x(n) = x(n-1) + \alpha * fVal / cVal, \quad \alpha \sim U(0,1)$$

Input:

- fVal : F(x), as N x 1 double.
- cVal : meritFunction(F(x)), as a scalar double.
- x : As a N x 1 double.
- nBees : The number of bees.

Output:

- draws : A double with size N x nBees with the randomly selected values of x around the current value.

See also:

[nb_beeSolver.initialize](#), [nb_beeSolver.scout](#)

Written by Kenneth Sæterhagen Paulsen

► evaluate ↑

```
obj = evaluate(obj,F,meritFunction)
```

Description:

Evaluate the bees at their tested locations.

Input:

- obj : A vector of nb_beeSolver objects.
- F : See the property with the same name in the nb_abcSolve class.
- meritFunction : See the property with the same name in the nb_abcSolve class.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_abcSolve.doSolving](#)

Written by Kenneth Sæterhagen Paulsen

► **evaluateParallel ↑**

obj = evaluateParallel(obj,F,meritFunction)

Description:

Evaluate the bees at their tested locations.

Input:

- obj : A vector of nb_beeSolver objects.
- F : See the property with the same name in the nb_abcSolve class.
- meritFunction : See the property with the same name in the nb_abcSolve class.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_abcSolve.doSolving](#)

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
[obj,funEvals] = initialize(obj,F,meritFunction,initialXValue,...  
    lowerBound,upperBound,objectiveLimit,maxTrials,...  
    local,newtonShare)
```

Description:

Initialize the bees that are going to be employed.

Input:

- obj : A vector of nb_beeSolver objects.
- See the properties with the same name in the nb_abcSolve class for help on the rest of the inputs.

Output:

- obj : A vector of nb_beeSolver objects. The type property has been set to 'employed'.
- funEvals : Number of function evaluations during initialization.

Written by Kenneth Sæterhagen Paulsen

► **joinBestDancer** ↑

```
joiners = joinBestDancer(joiners,minXValue,minFunctionValue,...  
    minFFunctionValue)
```

Description:

When we use the local option, the employed bees that runs out of nectar (exceeds the cutLimit) goes to the best nectar location of all time.

Input:

- joiners : A vector of nb_beeSolver objects.
- Otherwise see the properties with the same names in the nb_abcSolve class.

Output:

- joiners : A vector of nb_beeSolver objects.

See also:

[nb_beeSolver.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **joinDance** ↑

```
obj = joinDance(obj,employed,lowerBound,upperBound,local)
```

Description:

Make the onlooker or scout bees find a employed bee to dance with.

Input:

- obj : A vector of onlookers bees, as a vector of nb_beeSolver objects.
 - See the properties with the same names in the nb_abcSolve class.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_beeSolver.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **relocate** ↑

```
obj = relocate(obj,lowerBound,upperBound,cutLimit,local,...  
              F,jacobianFunction,tolerance,newtonStop,minXValue,...  
              minFunctionValue,minFFunctionValue)
```

Description:

Relocate the bees.

Input:

- obj : A vector of nb_beeSolver objects.
- Otherwise see the properties with the same names in the nb_abcSolve class.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_abcSolve.doSolving](#)

Written by Kenneth Sæterhagen Paulsen

► **scout** ↑

obj = scout(obj,lowerBound,upperBound)

Description:

Employ scouts to new dancing areas.

Input:

- obj : A vector of nb_beeSolver objects.
- lowerBound : See the property with the same name in the nb_abcSolve class.
- upperBound : See the property with the same name in the nb_abcSolve class.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_beeSolver.relocate](#)

Written by Kenneth Sæterhagen Paulsen

► **separate** ↑

[employed,onlookers] = separate(obj,cutLimit,tolerance,newtonStop)
[employed,onlookers,scouts] = separate(obj,cutLimit,tolerance,newtonStop)

Description:

Separate the bees into employed, scouts and onlookers.

Input:

- obj : A vector of nb_beeSolver objects.
- cutLimit : See the property with the same name in the nb_abcSolve class. Only needed if nargout > 2.

Output:

- employed : These are the employed bees that are keep dancing in their current area if nargout == 3, else it is all the employed bees. As a vector of nb_beeSolver objects.
- onlookers : These are the bees that are joing the dance of the most succesful employed bees. As a vector of nb_beeSolver objects.
- scouts : These are the employed bees that are going to scout for a new dancing area. As a vector of nb_beeSolver objects.

Written by Kenneth SÃ¶terhagen Paulsen

► **tournamentSelection** ↑

```
ret = tournamentSelection(obj,currentLeader)
```

Description:

Play out pairwise tournament between two bees.

Input:

- obj : The tested bee.
- currentLeader : The bee that is currently the best.

Output:

- ret : If true is returned tested be should be made the new leader.

See also:

[nb_beeSolver.updateLocation](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **updateLocation** ↑

```
obj = updateLocation(obj)
```

Description:

Update the location based on their last evaluation.

Input:

- obj : A vector of nb_beeSolver objects.

Output:

- obj : A vector of nb_beeSolver objects.

See also:

[nb_abcSolve.doSolving](#)

Written by Kenneth Sæterhagen Paulsen

■ nb_solve

Go to: [Properties](#) | [Methods](#)

A class for solving non-linear problems on the form

$F(x) = 0$

Constructor:

```
obj = nb_solve(fh,init)
obj = nb_solve(F,init,options,JF,varargin)
```

Input:

- F : See the documentation of the F property of the nb_solve class. Must be given.
- init : See the documentation of the init property of the nb_solve class. Must be given.
- options : See the nb_solve.optimset method for more on this input. Can be empty.
- JF : See the documentation of the JF property of the nb_solve class. Can be empty.

Optional inputs:

- varargin : Optional number of inputs given to the F (and JF) function(s) when it is called during the solution algorithm.

Output:

- obj : An object of nb_solve

Examples:

```
obj      = nb_solve(@(x)x^2,1);
[x,fVal] = nb_solve.call(@(x)x^2,1)
```

See also:

[fsolve](#), [nb_solve.call](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- | | | | |
|--------------------------------|-------------------------------|---------------------------------|--------------------------------------|
| • F | • JF | • alphaMethod | • criteria |
| • display | • display | • exitFlag | • fVal |
| • funEvals | • gamma | • initialXValue | • iter |
| • jacobian | • maxFunEvals | • maxIter | • maxIterSinceUpdate |
| • maxTime | • memory | • meritFVal | • meritFunction |
| • meritXChange | • method | • numWorkers | • options |
| • stepLength | • tolerance | • useParallel | • x |

- [F](#) [↑](#)

A function handle with the problem to solve. It is assumed to take a double with size M x 1 as input.

- [JF](#) [↑](#)

A function handle that return the jacobian of the problem to solve (F). It is assumed to take a double with size M x 1 as input. Only needed for the methods that uses derivative based methods.

- [alphaMethod](#) [↑](#)

Select the method to use to update the spectral step length when using the methods 'sane' or 'dfsane'. Choose amoung 1,2 or 3. Default is 2. See the methods [nb_solve.alphaMethod1](#), [nb_solve.alphaMethod2](#) and [nb_solve.alphaMethod3](#).

- [criteria](#) [↑](#)

Convergence criteria to use. Either 'function' (default) or 'stepSize'. 'function' means that the criteria being used is the first norm of F(x), while 'stepSize' means that the first norm of the step size (x(n) - x(n-1)) is used. See the property tolerance for the convergence measure.

- [display](#) [↑](#)

'off' | 'iter' | 'final' | 'notify' Level of display. 'off' displays no output; 'iter' displays output at each iteration 'final' displays just the final output; 'notify' displays output only if the function does not converge.

- [display](#) [↑](#)

An object of class nb_optimizerDisplayer. This object set the way the optimizer displays result during the optimization.

- **exitFlag** ↑

Reason for exiting the solving. A 1 means that the solver succeeded, while a value less than 1 means it failed. See the function nb_interpretExitFlag for how to interpret the exitFlag property.

- **fVal** ↑

Function value at last iteration, as M x 1 double. If the solver converged it will have norm less than given by the tolerance property.

- **funEvals** ↑

The number of function evaluations done during the solving of the problem.

- **gamma** ↑

Parameter used in the non-monotone line search step of the 'dfsane' and 'sane' methods.

- **initialXValue** ↑

Initial values. A M x 1 double.

- **iter** ↑

The number of iterations done during the solving of the problem.

- **jacobian** ↑

Jacobian matrix at last iteration, as M x M double.

- **maxFunEvals** ↑

Positive integer. Maximum number of function evaluation allowed. Default is inf.

- **maxIter** ↑

Positive integer. Maximum number of iterations allowed. Default is 10000.

- **maxIterSinceUpdate** ↑

Positive integer. Maximum number of iterations allowed between effective update. Default is 500. Used for global algorithm only.

- **maxTime** ↑

Positive integer. The maximum time spent on optimization in seconds.

- **memory** ↑

Positive integer. Set the number of periods that is remember in the 'dfsane' and 'sane' method. Normal to have a value between 5-20. Default is 10. High value will garanti a better approximation.

- **meritFVal** ↑

Merit value of $F(x)$. See meritFunction for the function applied. Default merit function is the first norm.

- **meritFunction** ↑

A function handle that specifies the merit function to use when convergence is checked. Default is $\ell_1(x)$. The merit function must map a $M \times 1$ double into a scalar double. A number close to 0 is taken as convergence.

- **meritXChange** ↑

Merit value of $x(n) - x(n-1)$. See meritFunction for the function applied. Default merit function is the first norm.

- **method** ↑

Select the method to use to solve the problem.

- > 'newton' : Uses a the Newton-Raphson method. The JF property must be given in this case.
- > 'steffensen' : Uses the Steffensen method. Where the derivative is numerically estimated.
- > 'steffensen2' : Uses a two steps steffensen method. Where the derivative is numerically estimated.
- > 'broyden' : Uses the Broyden method. Where the derivative is numerically updated at each iteration. Default method.
- > 'sane' : Uses the La Cruz and Raydan (2003) method.
- > 'dfsane' : Uses the La Cruz, Martínez, and Raydan (2006) method. Which is a derivativ free version of 'sane'.

- **numWorkers** ↑

Positive integer. Number of worker to use when running optimization in parallel.

- **options** ↑

Options of the solver selected. See the nb_solve.optimset method. If given as empty (struct) default options will be used.

- **stepLength** ↑

Step length applied when calculating numerical first difference objects. For example $(F(x + h) - F(x))/h$. h is here the selected step length

- **tolerance** ↑

Tolerance value for convergence. Default is 1e-7.

- **useParallel** ↑

true or false. Give true if the nodes should be spread on different parallel workers.

- **x** ↑

Solution at last iteration, as a M x 1 double. If the solver converged it will be the solution to the problem.

Methods:

- alphaMethod1
- alphaMethod2
- alphaMethod3
- broyden
- call
- dfsane
- get
- getMethods
- newton
- optimset
- sane
- set
- solve
- steffensen
- steffensen2
- test
- test2
- ypl

► **alphaMethod1** ↑

```
alpha = nb_solve.alphaMethod1(s,y)
```

Description:

Implements the spectral step length in equation 3 of Varadhan and Gilbert (2009).

```
alpha(k) = s(k-1)' * s(k-1) / ( s(k-1)' * y(k-1) )
```

See also:

[nb_solve.alphaMethod2](#), [nb_solve.alphaMethod3](#)

Written by Kenneth Sæterhagen Paulsen

► **alphaMethod2** ↑

```
alpha = nb_solve.alphaMethod2(s,y)
```

Description:

Implements the spectral step length in equation 4 of Varadhan and Gilbert (2009).

```
alpha(k) = s(k-1)' * s(k-1) / ( y(k-1)' * y(k-1) )
```

See also:

[nb_solve.alphaMethod1](#), [nb_solve.alphaMethod3](#)

Written by Kenneth Sæterhagen Paulsen

► **alphaMethod3** ↑

```
alpha = nb_solve.alphaMethod3(s,y)
```

Description:

Implements the spectral step length in equation 5 of Varadhan and Gilbert (2009).

```
alpha(k) = sgn(s(k-1)' * y(k-1)) * ||s(k-1)|| / ||y(k-1)||
```

where $\text{sgn}(x) = x/|x|$

See also:

[nb_solve.alphaMethod1](#), [nb_solve.alphaMethod2](#)

Written by Kenneth Sæterhagen Paulsen

► **broyden** ↑

```
results = nb_solve.broyden(opt,fVal)
```

Description:

Solve the problem using the Broyden algorithm.

See also:

[nb_solve.solve](#)

Written by Kenneth Sæterhagen Paulsen

► call ↑

```
[x,fval,exitflag,JAC] = nb_solve.call(F,init,options,JF,varargin)
```

Description:

Call the nb_solve class to solve the $F(x) =$ problem. See the documentation of the nb_abc class for more on this algorithm.

Input:

- F : See the documentation of the F property of the nb_solve class. Must be given.
- init : See the documentation of the init property of the nb_solve class. Must be given.
- options : See the nb_solve.optimset method for more on this input. Can be empty.
- JF : See the documentation of the JF property of the nb_solve class. Can be empty.

Optional inputs:

- varargin : Optional number of inputs given to the F (and JF) function(s) when it is called during the solution algorithm.

Output:

- x : See the documentation of the minXValue property of the nb_abc class.
- fval : See the documentation of the minFunctionValue property of the nb_abc class.
- exitflag : See the documentation of the exitFlag property of the nb_abc class.
- JAC : Jacobian at last iteration.

Examples:

```
g = @(x)x^2;
x1 = nb_solve.call(g,1)
x2 = nb_solve.call(g,-3)
```

See also:

[nb_solve.optimset](#), [fsolve](#)

Written by Kenneth Sætherhagen Paulsen

► **dfsane** ↑

```
results = nb_solve.dfsane(opt,fVal)
```

Description:

Solve the problem using the Spectral Residual Method of La Cruz W, Martínez JM, Raydan M (2006). The implementation follows Varadhan and Gilbert (2009) with some adjustments.

See also:

[nb_solve.solve](#)

Written by Kenneth Sæterhagen Paulsen

► **get** ↑

```
get(obj,propertyName)
```

Description:

Gets the property of an object of class `nb_getable`.

Input:

- `obj` : An object of class `nb_getable`.
- `propertyName` : Name of the property to get the value of, as a string.

Output:

- `value` : The value of the property asked for.

Examples:

```
fig = obj.get('parent');
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass `NB_GETABLE`

► **getMethods** ↑

```
methods = getMethods()
```

Description:

Get a list of the supported methods of the nb_solve class.

Output:

- methods : A cellstr with the supported methods of the nb_solve class.

See also:

[nb_solve.call](#), [nb_solve.optimset](#)

Written by Kenneth Sæterhagen Paulsen

► [newton](#) ↑

results = nb_solve.newton(opt,fVal)

Description:

Solve the problem using the Newton-Raphson algorithm.

See also:

[nb_solve.solve](#)

Written by Kenneth Sæterhagen Paulsen

► [optimset](#) ↑

options = nb_solve.optimset(varargin)

Description:

Get solution options or help.

Optional input:

- Run without inputs to get defaults.
- Run with 'list' as the first input to get a cellstr with the supported options.
- Run with 'help' to get a struct with help on each option.
- Run with 'optionName' to get help on a particular option, i.e. only one input.
- Use 'optionName', optionValue pairs to set options of the returned struct.

Output:

- Depends on the input.

See also:[nb_abc.options](#)

Written by Kenneth SÃ¶terhagen Paulsen

► sane ↑

```
results = nb_solve.sane(opt,fVal)
```

Description:

Solve the problem using the Spectral Residual Method of La Cruz and Raydan (2003). The implementation follows Varadhan and Gilbert (2009) with some adjustments.

See also:[nb_solve.solve](#)

Written by Kenneth SÃ¶terhagen Paulsen

► set ↑

```
set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_settable. Give the property name as a string. The input that follows should be the value you want to assign to that property.

Multple properties could be set with one method call.

Input:

- obj : An object of class nb_settable.
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_settable with the given properties reset.

Examples:

```
obj = obj.set('parent',figure);
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_SETTABLE

► **solve** ↑

```
solve(obj,varargin)
```

Description:

Solve the problem.

Input:

- obj : An object of class nb_solve.

See also:

[nb_solve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **steffensen** ↑

```
results = nb_solve.steffensen(opt,fVal)
```

Description:

Solve the problem using the Steffensen algorithm.

See also:

[nb_solve.solve](#)

Written by Kenneth Sæterhagen Paulsen

► **steffensen2** ↑

```
results = nb_solve.steffensen2(opt,fVal)
```

Description:

Solve the problem using a two step version of the Steffensen algorithm.

See also:

[nb_solve.solve](#)

Written by Kenneth Sæterhagen Paulsen

► **test** ↑

```
solver = nb_solve.test(varargin)
```

Description:

Solve the following problem:

$x^2 = 0$

Which has the trivial solution 0.

Optional input:

- varargin : Optional inputs given to the set method.

Output:

- solver : An object of class nb_solve.

See also:

[nb_solve](#), [nb_solve.call](#)

Written by Kenneth Sæterhagen Paulsen

► **test2** ↑

```
solver = nb_solve.test2(varargin)
```

Description:

Solve the following problem:

$x(1)^2 - x(2) = 0$
 $x(1) * x(2) = 0$

Which has the trivial solution [0;0].

Optional input:

- varargin : Optional inputs given to the set method.

Output:

- solver : An object of class nb_solve.

See also:

[nb_solve](#), [nb_solve.call](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **yp1** ↑

```
results = nb_solve.dfsane(opt,fVal)
```

Description:

Solve the problem using the projection method suggested by Yan, Peng and Li (2008).

See also:

[nb_solve.solve](#)

Written by Kenneth SÃ¶terhagen Paulsen

◆ **nb_blockDecompose**

```
blocks = nb_blockDecompose(F,y,p,tol,fixed,varargin)
```

Description:

Block decompose system of non-linear equations $F(y, p, \text{varargin}{:}) = 0$.

For more see see Mihoubi (2011), "Solving and estimating stochastic models with block decomposition" section 2.

Input:

- F : A function_handle that takes 2 inputs.
- y : The initial values of the variables to solve for, as a nVar x 1 double.
- p : The parameters of the function F (this may include exogenous variables!). As a nPar x 1 double.
- tol : The tolerance level for setting elements of the jacobian to 0.
- fixed : A 1 x nVar logical with elements set to true if the variable value is already known.

Optional input:

- varargin : Optional inputs given to the function handle F when being evaluated.

Output:

- blocks : A struct with how to block the model.

See also:

[nb_solver](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_getSolvers

solvers = nb_getSolvers()

Description:

Get the solvers that solves systems of nonlinear equations of several variables.

This function also tries to detect solvers of other toolboxes, such as csolve (Chris Sims). You need to add these optimimzer separately, as they are not part of NB toolbox.

Output:

- solvers : A cellstr with the supported solvers.

See also:

[fsolve](#), [nb_getDefaultOptimset](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_homotopy

```
x           = nb_homotopy(algorithm,steps,start,finish,...  
                           names,solveFunc,fHandle,options)  
[x,fVal,exitflag,err] = nb_homotopy(algorithm,steps,start,finish,...  
                           names,solveFunc,fHandle,init,options,varargin)
```

Description:

Find the solution of the problem $F(x) = 0$ using homotopy.

At each step of the homotopy algorithm the solver function (solveFunc) will be called.

Caution: It is assumed that the problem can be solved with the parameters in start with the given initial conditions.

Input:

- algorithm : Either 1, 2 or 3.
 - > 1: The problem is solved iteratively. Starting with the vector of parameters in start until it reaches the parameter vector finish. All the parameters will change at the same time. This means that the problem is solved steps times.
 - > 2: Same as 1, but know the only one parameter is change at each recursion. One step of this algorithm is when all the parameters have changed. This means that the problem is solved nParam x steps times.
 - > 3: Now it tries to solve the problem at finish with the solution from the parameters in start as the initial condition. If this fails, it will try to solve it with the parameter that are the mid point between start and finish. If that fails, it splits this interval in two again, and it will continue in this way until it finds a point it can solve the problem. It will do this for each sub interval until it reaches the parameter vector finish.
- steps : Depends on the algorithm input.
 - > 1: The number of recursive steps from the start vector of parameters until the finish vector of parameter.
 - > 2: The number of recursive steps from the start vector of parameters until the finish vector of parameter.
 - > 3: The maximum number of allowed steps of this algorithm.
- start : The nParam x 1 parameter vector at the starting point, where we assume we already know that the problem can be solved.
- finish : The nParam x 1 parameter vector at the finishing point. This point is the point where we are really interested in solving the problem.
- names : A cellstr with the names of the parameters to do the homotopy over. If given as empty default names will be given. This is to throw a correct error message when homotopy algorithm 2 fails.
- solveFunc : See the same input to the nb_solver function.
- fHandle : See the same input to the nb_solver function.
- init : See the same input to the nb_solver function.
- options : See the same input to the nb_solver function. To prevent it from printing iteration on the command line set the silent field to true.

Optional input:

- varargin : See the same input to the nb_solver function.

Output:

```
- x      : The solution of the problem F(x) = 0, as a vector of double values.

- fVal   : The value of F(x) at the return x.

- exitflag : The exit flag return by the solver. If only to outputs are returned this exit flag will be interpreted by the nb_interpretExitFlag and an error will be thrown by inside this function.

- err     : A struct with the information on the error during the homotopy steps. Empty if no errors occur. This can be given to the nb_interpretExitFlag function to throw a generic error message. The fields are:

  > iter    : If exitflag return that the problem could not be solve, this output will give the iteration number it first failed. Otherwise it is empty. For algorithm 2 this will be a 1x2 double, where the 2 element is the parameter index it failed.

  > points  : If exitflag return that the problem could not be solve, this output will give the parameter vector it first failed.

  > algorithm : Same as the algorithm input.

  > names    : Same as the names input.
```

Written by Kenneth Sætherhagen Paulsen

◆ nb_lyapunovEquation

```
[X,failed] = nb_lyapunovEquation(A,B,tol,maxiter)
```

Description:

Solve the problem $X = A \cdot X \cdot A' + B$ using the fact that:

```
X = sum A^i * B * (A')^i over i = 1:inf
```

This method iterate the sum above until convergence.

Input:

```
- A      : A symmetric matrix with size N x N

- B      : A symmetric matrix with size N x N

- tol    : The tolerance of the iteration procedure. Default is eps^(1/3).

- maxiter : Maximum number of iterations. Default is 1000.
```

Output:

- X : Solution of the problem. As a N x N double.
- failed : True if the calculation failed.

Written by Kenneth Sæterhagen Paulsen

◆ nb_lyapunovEquation2

```
[X,failed] = nb_lyapunovEquation2(A,B,C,tol,maxiter)
```

Description:

Solve the problem $X = A \cdot X \cdot B + C$ using the fact that:

$X = \sum A^i \cdot C \cdot B^i$ over $i = 1:\inf$

This method iterate the sum above until convergence.

Input:

- A : A symmetric matrix with size N x N
- B : A symmetric matrix with size N x N
- C : A symmetric matrix with size N x N
- tol : The tolerance of the iteration procedure. Default is $\text{eps}^{(1/3)}$.
- maxiter : Maximum number of iterations. Default is 1000.

Output:

- X : Solution of the problem. As a N x N double.
- failed : True if the calculation failed.

Written by Kenneth Sæterhagen Paulsen

◆ nb_solveLinearRationalExpModel

```
[D,L,err] = nb_solveLinearRationalExpModel(A,B,nState)
```

Description:

Solves for the recursive representation of the stable solution to a system of linear difference equations.

Klein, Paul, 2000. "Using the generalized Schur form to solve a multivariate linear rational expectations model," Journal of Economic

Dynamics and Control, Elsevier, vol. 24(10), pages 1405–1423, September.

Solves $A[x(t+1)] = B x(t)$. $x(t)$ is ordered so that the state variables comes first.

Inputs:

- A, B : Two square matrices A and B in the equation above.
- nState : Number of state variables.

Outputs:

- D, L : The decision rule D and the law of motion L. Lets decompose $x(t)$ into $x(t) = [s(t); c(t)]$, where $s(t)$ are the state variables. Then the solution can be found to be:

$$\begin{aligned}c(t) &= D*s(t) \text{ and} \\s(t+1) &= L*s(t).\end{aligned}$$

- err : Non-empty if either the Blanchard-Kahn rank condition or order condition is not satisfied.

See also:

[qz](#), [ordqz](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_solver**

```
x = nb_solver(solveFunc,fHandle,init,options)
[x,fVal,exitflag] = nb_solver(solveFunc,fHandle,init,options,varargin)
```

Description:

Call a solver function (solveFunc) that can be called like

```
[x,fVal,exitflag] = solveFunc(fHandle,init,options,varargin{:})
```

The function should solve the problem $F(x) = 0$ for x .

Input:

- solveFunc : A string with the name or a function_handle of the function to use to solve the problem
- fHandle : A function handle that represents F.
- init : The initial values of x to be used by the solver.
- options : A struct with the options to use by the solver. For some optimizers see the optimset function by MATLAB.

Options input:

- varargin : These inputs is passed on to the function fHandle during the solving of the problem.

Output:

- x : The solution of the problem $F(x) = 0$, as a vector of double values.
- fVal : The value of $F(x)$ at the return x.
- exitflag : The exit flag return by the solver. If only to outputs are returned this exit flag will be interpreted by the nb_interpretExitFlag and an error will be thrown by inside this function.

See also:

[fsolve](#), [nb_getDefaultOptimset](#), [nb_getSolvers](#)

Written by Kenneth Sæterhagen Paulsen

25.22 Utils

- gaimc.bfs
- gaimc.csr_to_sparse
- gaimc.sparse_to_csr
- nb_bartlettKernel
- nb_bgrowth
- nb_cell2func
- nb_constructInputPar
- nb_cov2corr
- nb_doSymbolic
- nb_equation
- nb_funcWrapper
- nb_getModelListFromLibrary
- nb_gradient
- nb_interpretPerc
- nb_jacobian
- nb_loadModelsFromLibrary
- nb_logger
- nb_maxDeriv
- nb_monteCarloSim
- nb_num
- nb_parzenKernel
- nb_shortestpath
- nb_stTerm
- nb_symMatrix
- nb_varOrPar
- nb_writePostMacroFile
- gaimc.convert_sparse
- gaimc.dijkstra
- myAD
- nb_base
- nb_calculateMoments
- nb_coeff2excel
- nb_constructStackedCorrelationMatrix
- nb_createGenericNames
- nb_eq2Latex
- nb_excel2coeff
- nb_functionHandle2Struct
- nb_getSymbolicDerivIndex
- nb_hessian
- nb_isStructOfFunctionHandle
- nb_latinHypercubeSim
- nb_logLinearize
- nb_macro
- nb_minDeriv
- nb_mySD
- nb_param
- nb_quadraticSpectralKernel
- nb_st
- nb_struct2functionHandle
- nb_testForecastRestrictions
- nb_writeHelp

► **gaimc.bfs** ↑

```
[d, dt, pred] = gaimc.bfs(A,u,target)
```

Description:

Compute breadth first search distances, times, and tree for a graph

```
[d, dt, pred] = bgl.bfs(A,u) returns the distance (d) and the discover time (dt) for each vertex in the graph in a breadth first search starting from vertex u.  
d = dt(i) = inf if vertex i is not reachable from u  
pred is the predecessor array. pred(i) = 0 if vertex (i) is in a component not reachable from u and i != u.  
[...] = bgl.bfs(A,u,v) stops the bfs when it hits the vertex v
```

Examples:

```
W = ones(1,11);  
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W);  
[dist,dt,pred] = gaimc.bfs(DG,1)
```

See also:

[gaimc.dijkstra](#), [nb_shortestpath](#)

Written by David F. Gleich

Edited by Kenneth S. Paulsen

- Return inf instead of -1 if vertex is not reachable from u.

► **gaimc.convert_sparse** ↑

```
As = gaimc.convert_sparse(A)
```

Description:

Convert a sparse matrix to the native gaimc representation

As = convert_sparse(A) returns a struct with the three arrays defining the compressed sparse row structure of A.

Examples:

```
load('graphs/all_shortest_paths_example')  
As = bgl.convert_sparse(A)
```

See also:

[gaimc.sparse_to_csr](#), [sparse](#)

Written by David F. Gleich

► **gaimc.csr_to_sparse** ↑

```
[nzi,nzj,nzv] = gaimc.csr_to_sparse(rp,ci,ai,ncols)
```

Description:

Convert from compressed row arrays to a sparse matrix.

A = gaimc.csr_to_sparse(rp,ci,ai) returns the sparse matrix represented by the compressed sparse row representation rp, ci, and ai. The number of columns of the output sparse matrix is max(max(ci),nrows). See the call below.

A = gaimc.csr_to_sparse(rp,ci,ai,ncol) While we can infer the number of rows in the matrix from this expression, you may want a different number of.

[nzi,nzj,nzv] = gaimc.csr_to_sparse(...) returns the arrays that feed the sparse call in matlab. You can use this to avoid the sparse call and customize the behavior.

This command "inverts" the behavior of gaimc.sparse_to_csr.
Repeated entries in the matrix are summed, just like sparse does.

Examples:

```
A          = sparse(6,6);
A(1,1)    = 5;
A(1,5)    = 2;
A(2,3)    = -1;
A(4,1)    = 1;
A(5,6)    = 1;
[rp, ci, ai] = gaimc.sparse_to_csr(A);
A2         = gaimc.csr_to_sparse(rp,ci,ai)
```

See also:

[gaimc.sparse_to_csr](#), [sparse](#)

Written by David F. Gleich

► **gaimc.dijkstra** ↑

```
[d,dt,pred,m] = gaimc.dijkstra(A,u)
```

Description:

Compute shortest paths using Dijkstra's algorithm

d = gaimc.dijkstra(A,u) computes the shortest path from vertex u to all nodes reachable from vertex u using Dijkstra's algorithm for the problem. The graph is given by the weighted sparse matrix A, where A(i,j) is the distance between vertex i and j. In the output vector d, the entry d(v) is the minimum distance between vertex u and vertex v.

A vertex w unreachable from u has d(w)=Inf.

[d,dt,pred,m] = gaimc.dijkstra(A,u) also returns the discover time (dt) for each vertex in the graph and the predecessor tree to generate the actual shortest paths. In the predecessor tree pred(v) is the vertex preceding v in the shortest path and pred(u)=0. Any unreachable vertex has pred(w)=0 as well. m is the number of nodes along the path.

If your network is unweighted, then use bfs instead.

See also BFS

Examples:

```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W);
[dist,dt,pred] = gaimc.dijkstra(DG,1)
```

See also:

[gaimc.bfs](#), [nb_shortestpath](#)

Written by David F. Gleich

Edited by Kenneth S. Paulsen
- Added the m output.

► **[gaimc.sparse_to_csr](#)** ↑

```
[rp,ci,ai,ncol] = gaimc.sparse_to_csr(A,varargin)
```

Description:

Convert a sparse matrix into compressed row storage arrays

[rp, ci, ai] = gaimc.sparse_to_csr(A) returns the row pointer (rp), column index (ci) and value index (ai) arrays of a compressed sparse representation of the matrix A.

[rp ci ai] = gaimc.sparse_to_csr(i,j,v,n) returns a csr representation of the index sets i,j,v with n rows.

Example:

```
A = sparse(6,6); A(1,1)=5; A(1,5)=2; A(2,3)=-1; A(4,1)=1; A(5,6)=1;
[rp, ci, ai] = gaimc.sparse_to_csr(A)
```

See also:

[gaimc.csr_to_sparse](#), [sparse](#)

Written by David F. Gleich

■ myAD

Go to: [Properties](#) | [Methods](#)

```
/*
 * myAD and myA2D - Automatic Differentiation of 1st
 * Copyright (C) 2006 Martin Fink. (martinfink "at" gmx.at)
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, visit
 * http://www.gnu.org/licenses/gpl.html or write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
 */
License. This is released with GPL 2.1.
```

Properties:

Methods:

- | | | | | |
|----------------------------|--------------------------------|--------------------------------------|-----------------------------|-----------------------------|
| • abs | • acos | • asin | • atan | • circshift |
| • cos | • ctranspose | • cumprod | • cumsum | • diff |
| • disp | • end | • eq | • exp | • find |
| • fsolve | • ge | • get | • getderivs | • getvalues |
| • gt | • horzcat | • isnan | • ldivide | • le |
| • length | • log | • logncdf | • lognpdf | • lt |
| • max | • min | • minus | • mldivide | • mpower |
| • mrdivide | • mtimes | • ne | • normcdf | • norminv |
| • normpdf | • numel | • permute | • plus | • power |
| • prod | • rdivide | • repmat | • reshape | • sin |
| • sinh | • size | • sort | • sparse | • spdiags |
| • sqrt | • steady_state | • steady_state_first | • subsasgn | • subsref |
| • sum | • tan | • tanh | • times | • transpose |
| • uminus | • uplus | • vertcat | • zeros | |

► **abs** ↑

x = abs(x)

Description:

Absolute value.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD – Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **acos** ↑

$x = \text{acos}(x)$

Description:

Inverse cosine in radians.

Edited by SeHyoun Ahn, May 2016

In Package myAD – Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **asin** ↑

$x = \text{asin}(x)$

Description:

Inverse sine in radians.

Edited by SeHyoun Ahn, May 2016

In Package myAD – Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **atan** ↑

$x = \text{atan}(x)$

Description:

Inverse tangent in radians.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **circshift** ↑

x = circshift(x,varargin)

Description:

Missing.

Written by SeHyoun Ahn, July 2016

► **cos** ↑

x = acos(x)

Description:

Cosine.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **ctranspose** ↑

x = ctranspose(x)

Description:

Transpose. Only allows real derivatives, so this does not conjugate derivative values

Written by SeHyoun Ahn, Jan 2016

► **cumprod** ↑

x = cumprod(x,varargin)

Description:

Cumulative product.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **cumsum** ↑

x = cumsum(x,varargin)

Description:

Cumulative sum.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **diff** ↑

x = diff(x,varargin)

Description:

Difference.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **disp** ↑

x = disp(x,varargin)

Description:

Display object in commando line.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **end ↑**

`x = end(x,varargin)`

Description:

end operator.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **eq ↑**

`x = eq(x,y)`

Description:

Test for equality of values.

This only checks if the values are equal. There is no guarantee that the derivatives are equal.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **exp ↑**

`x = exp(x)`

Description:

Exponential.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **find** ↑

```
[i,j,v] = find(A)
```

Description:

Find.

This function does not support all use cases of find.

Written by SeHyoun Ahn, Oct 2017

► **fsolve** ↑

```
x = fsolve(h,x0,varargin)
```

Description:

Check included README.pdf to see the syntax for vector valued fsolve.

Written by SeHyoun Ahn, Jan 2016

► **ge** ↑

```
x = ge(x,y)
```

Description:

Greater than or equal. Only tests the values not derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **get** ↑

```
x = get(x,varargin)
```

Description:

Get value of property.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **getderivs** ↑

```
Jac = getderivs(x)
```

Description:

Get evaluated derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **getvalues** ↑

```
x = getvalues(x)
```

Description:

Get evaluated values.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **gt** ↑

```
x = ge(x,y)
```

Description:

Greater than. Only tests the values not derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **horzcat** ↑

```
x = horzcat(varargin)
```

Description:

Horizontal concatenation.

Written by SeHyoun Ahn, Jan 2016

► **isnan** ↑

```
x = isnan(x)
```

Description:

Test for nan in values or derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **ldivide** ↑

```
x = ldivide(x,y)
```

Description:

Same as rdivide.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **le** ↑

```
x = le(x,y)
```

Description:

Less than or equal. Only tests the values not derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **length** ↑

```
mylength = length(x)
```

Description:

length operator.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **log** ↑

```
x = isnan(x)
```

Description:

Natural logarithm.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **logncdf** ↑

```
x = logncdf(x,m,k)
```

Description:

Log normal cdf.

Written by Kenneth Sæterhagen Paulsen

► **lognpdf** ↑

```
x = lognpdf(x,m,k)
```

Description:

Log normal pdf.

Written by Kenneth SÅ!terhagen Paulsen

► **lt** ↑

x = ge(x,y)

Description:

Less than. Only tests the values not derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **max** ↑

varargout = max(x,varargin)

Description:

Max operator.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **min** ↑

varargout = min(x,varargin)

Description:

Min operator.

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **minus** ↑

```
x = minus(x,y)
```

Description:

Minus.

Edited by SeHyoun Ahn, Oct 2017

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **mldivide** ↑

```
y = mldivide(x,y)
```

Description:

Matrix left division (\).

Written by SeHyoun Ahn, Jan 2016

► **mpower** ↑

```
x = plus(x,y)
```

Description:

Redirect to power (.^) if x is a scalar.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **mrdivide** ↑

```
y = mldivide(x,y)
```

Description:

Matrix right division (/).

Written by SeHyoun Ahn, Jan 2016

► **mtimes** ↑

```
y = mldivide(x,y)
```

Description:

Matrix multiplication.

Note that in the original package by Martin Fink, this redirected to element-wise multiplication

Written by SeHyoun Ahn, Jan 2016

► **ne** ↑

```
x = ne(x,y)
```

Description:

Not equal. Only tests the values not derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **normcdf** ↑

```
x = normcdf(x,m,k)
```

Description:

Normal cdf.

Written by Kenneth Sæterhagen Paulsen

► **norminv** ↑

```
x = norminv(x,m,k)
```

Description:

Inverse normal cdf.

Written by SeHyoun Ahn, May 2022
Edited by Kenneth S. Paulsen
- Added the m and k inputs

► **normpdf** ↑

```
x = normcdf(x,m,k)
```

Description:

Normal cdf.

Written by Kenneth Sæterhagen Paulsen

► **numel** ↑

```
x = numel(x)
```

Description:

The numel operator.

Written by SeHyoun Ahn, July 2016

► **permute** ↑

```
x = permute(x,order)
```

Description:

Permute matrix operator.

Written by SeHyoun Ahn, July 2016

► **plus** ↑

```
x = plus(x,y)
```

Description:

Plus (+).

Edited by SeHyoun Ahn, Oct 2017

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **power** ↑

x = plus(x,y)

Description:

power (.^).

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **prod** ↑

x = prod(x,varargin)

Description:

Product.

Written by SeHyoun Ahn, July 2016

► **rdivide** ↑

x = rdivide(x,y)

Description:

Right division (./).

Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **repmat** ↑

x = repmat(x,varargin)

Description:

Replicate object.

Written by SeHyoun Ahn, Jan 2016

► **reshape** ↑

```
x = reshape(x,varargin)
```

Description:

Reshape object.

Written by SeHyoun Ahn, Jan 2016

► **sin** ↑

```
x = sin(x)
```

Description:

Sine.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, June 2006
martinfink 'at' gmx.at

► **sinh** ↑

```
x = sin(x)
```

Description:

Hyperbolic sine.

Written by SeHyoun Ahn, Jan 2016

► **size** ↑

```
x = size(x,varargin)
```

Description:

Size of object.

Written by SeHyoun Ahn, July 2016

► **sort ↑**

```
varargout = sort(x,varargin)
```

Description:

Sort based on the values.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **sparse ↑**

```
A = sparse(i,j,v,n,m)
```

Description:

Make object sparse.

Written by SeHyoun Ahn, Oct 2017

► **spdiags ↑**

```
x = spdiags(x,varargin)
```

Description:

Note that spdiags will keep the derivative matrix in column-major order %
Therefore, given matrix A(i,j), the first few rows of the derivative
matrices are

```
D_x A_{1,1} (These are row vectors of derivative of A_{1,1})  
D_x A_{2,1}  
...  
D_x A_{n,1}  
D_x A_{1,2}  
...
```

Written by SeHyoun Ahn, Jan 2016

► **sqrt** ↑

```
x = isnan(x)
```

Description:

Test for nan in values or derivatives.

Edited by SeHyoun Ahn, May 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **steady_state** ↑

```
x = steady_state(x)
```

Description:

Derivative of the steady_state operator. Used by nb_DSGE. Will return a double.

Written by Kenneth Sæterhagen Paulsen

► **steady_state_first** ↑

```
x = steady_state_first(x)
```

Description:

Derivative of the steady_state_first operator. Used by nb_DSGE. Will return a double.

Written by Kenneth Sæterhagen Paulsen

► **subsasgn** ↑

```
y = subsasgn(y, S, x)
```

Description:

Subs asgment of object.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **subsref** ↑

x = subsref(x, S)

Description:

Subs reference object.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **sum** ↑

x = sum(x,varargin)

Description:

Sum.

Edited by SeHyoun Ahn, July 2016
Edited by SeHyoun Ahn, Jan 2016

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **tan** ↑

x = tan(x)

Description:

Tangent.

Edited by SeHyoun Ahn, May 2016

In Package myAD – Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **tanh** ↑

`x = tanh(x)`

Description:

Hyperbolic tangent.

Edited by SeHyoun Ahn, May 2016

In Package myAD – Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **times** ↑

`x = times(x, varargin)`

Description:

Multiplication ($\cdot\ast$).

Edited by SeHyoun Ahn, Jan 2016

► **transpose** ↑

`x = transpose(x)`

Description:

Transpose.

Written by SeHyoun Ahn, Jan 2016

► **uminus** ↑

```
x = uminus(x)
```

Description:

Unary minus.

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **uplus** ↑

```
x = uplus(x)
```

Description:

Unary plus.

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

► **vertcat** ↑

```
x = vertcat(varargin)
```

Description:

Vertical concatenation.

Written by SeHyoun Ahn, Jan 2016

► **zeros** ↑

```
x = zeros(x)
```

Description:

Zero out all values and derivatives.

In Package myAD - Automatic Differentiation
by Martin Fink, May 2007
martinfink 'at' gmx.at

■ nb_base

Go to: [Properties](#) | [Methods](#)

A class for representing a variable. Used by the nb_term class.

Superclasses:

nb_term

Constructor:

obj = nb_base(value,sign);

Input:

- value : A variable name. E.g. 'x'.

Output:

- obj : An object of class nb_base.

Examples:

```
base = nb_base('x');
```

See also:

[nb_term](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [numberOfTerms](#)
- [operator](#)
- [terms](#)
- [value](#)

- [numberOfTerms](#) ↑

The number of terms. As a scalar integer.

Inherited from superclass NB_TERM

- [operator](#) ↑

Type of operator splitting the terms.

Inherited from superclass NB_TERM

- [terms](#) ↑

A vector of nb_term objects.

Inherited from superclass NB_TERM

- [value](#) ↑

Base value, as a one line char.

Methods:

- [callExpOnSub](#)
- [callFuncOnSub](#)
- [cellstr](#)
- [clean](#)
- [correct](#)
- [eq](#)
- [exp](#)
- [gap](#)
- [generalFunc](#)
- [initialize](#)
- [intersect](#)
- [ismember](#)
- [log](#)
- [logncdf](#)
- [lognpdf](#)
- [minus](#)
- [mpower](#)
- [mrdivide](#)
- [mtimes](#)
- [normcdf](#)
- [norminv](#)
- [normpdf](#)
- [plus](#)
- [power](#)
- [rdivide](#)
- [removePar](#)
- [simplify](#)
- [split](#)
- [sqrt](#)
- [steady_state](#)
- [times](#)
- [toString](#)
- [uminus](#)
- [union](#)
- [uplus](#)

► **callExpOnSub** ↑

```
obj = callExpOnSub(obj)
```

See also:

[nb_term.exp](#)

Written by Kenneth Sæterhagen Paulsen

► **callFuncOnSub** ↑

```
obj = callFuncOnSub(obj, func, varargin)
```

See also:

[nb_term.generalFunc](#)

Written by Kenneth Sæterhagen Paulsen

► **cellstr** ↑

```
c = toString(obj)
```

Description:

Convert array of nb_term objects to cellstr.

Inputs:

- obj : An array of nb_term objects.

Output:

- c : A cellstr with same size as obj.

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **clean** ↑

obj = clean(obj)

Description:

Clean up nb_term object after operations.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **correct** ↑

expr = nb_term.correct(expr)

Description:

Correct term for .*, ./ and .^ operators.

Input:

- expr : A one line char with a mathematical expression.

Output:

- expr : Corrected version of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **eq** ↑

test = eq(obj,another)

Description:

Test for equality of two objects. They are only equal if the value property is equal.

Inputs:

- obj : A nb_base object.
- another : A nb_base object.

Output:

- test : true or false.

Written by Kenneth Sæterhagen Paulsen

► **exp ↑**

obj = exp(obj)

Description:

exp operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **gap ↑**

obj = gap(obj)

Description:

Gap operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► generalFunc ↑

```
obj = generalFunc(obj,func,varargin)
```

Description:

Call general function on a nb_term objects.

Input:

- obj : A vector of nb_term objects.
- func : A one line char with the function to apply on the nb_term objects.

Optional input:

- nb_term objects representing the extra inputs to the function.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► initialize ↑

```
obj = nb_term.initialize(rows,cols)
```

Description:

Initialize vector of nb_term objects.

Input:

- rows : Number of rows. As a scalar integer.
- cols : Number of columns. As a scalar integer.

Output:

- obj : A rows x cols nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► intersect ↑

```
int = intersect(obj,another)
```

Description:

Get the intersect of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- int : The intersect, as a vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► ismember ↑

```
[ind,loc] = ismember(obj,another)
```

Description:

Check if the terms in obj is member of the terms in another.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- ind : A logical vector with same size as obj. Elements that are true are found to be in the another input.
- loc : A double vector with same size as obj. Locations of the

elements that are found in the another input. If not found 0 is returned.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **log** ↑

obj = log(obj)

Description:

log operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **logncdf** ↑

obj = logncdf(obj,m,k)

Description:

Log-normal cdf.

Input:

- obj : A vector of nb_term objects.

- m : The mean of the distribution, as an object of class nb_term.

- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **lognpdf** ↑

```
obj = lognpdf(obj,m,k)
```

Description:

Log-normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **minus** ↑

```
obj = minus(obj,another)
```

Description:

Minus operator (-) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mpower** ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mrdivide** ↑

```
obj = mrdivide(obj,another)
```

Description:

Division operator (/) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► mtimes ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► normcdf ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► norminv ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► normpdf ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► plus ↑

obj = plus(obj,another)

Description:

Plus operator (+) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► power ↑

obj = power(obj,another)

Description:

Power operator (.^) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **rdivide** ↑

obj = rdivide(obj,another)

Description:

Division operator (./) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **removePar** ↑

expr = removePar(expr)

Description:

Is the expression enclosed with parentheses?

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **simplify** ↑

```
expr = nb_term.simplify(expr)
expr = nb_term.simplify(expr,waitbar)
```

Description:

Simplify the mathematical expressions.

Input:

- expr : A one line char with a mathematical expression or a cellstr with the mathematical expressions.
- waitbar : A nb_waitbar5 object. It will use the next available waitbar of the given object. If empty no waitbar is added.

Output:

- obj : A cellstr with the simplified mathematical expressions.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **split** ↑

```
obj = nb_term.split(expr)
obj = nb_term.split(expr,waitbar)
```

Description:

Split the mathematical expression into separate terms.

Input:

- expr : A one line char with a mathematical expression.
- waitbar : A nb_waitbar5 object. If empty no waitbar is added.

Output:

- obj : A nb_term object representing the mathematical expression.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

sqrt operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **steady_state** ↑

```
obj = steady_state(obj)
```

Description:

Steady-state operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **times** ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► [toString](#) ↑

c = toString(obj)

Description:

Convert nb_base object to string or cellstr.

Inputs:

- obj : A nb_base object. May also be a matrix.

Output:

- c : If obj is scalar the output will be a one line char, otherwise it will be a cellstr with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► [uminus](#) ↑

obj = uminus(obj)

Description:

Unary minus operator (-) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **union** ↑

uni = union(obj,another)

Description:

Get the union of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.

- another : A vector of nb_term objects.

Output:

- uni : The union, as a vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **unique** ↑

uniq = unique(obj,another)

Description:

Locate the unique nb_term objects in a vector of nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- uniq : The unique nb_term objects.

- ind : Vector such that uniq = obj(ind).

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus opertor (+) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_TERM

■ **nb_bgrowth**

Go to: [Properties](#) | [Methods](#)

A class for deriving the balanced growth restrictions of a DSGE model.

Constructor:

```
obj = nb_bgrowth(varName,constant,precision);
```

Input:

- varName : Name of the variable(s) to take derivative respect to, either as a char or cellstr.

- constant : Set to true if this object represent a stationary variable or parameter.

- precision : See help on the property precision.

Output:

- obj : An object of class nb_bgrowth

See also:

[nb_st](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- constant
- equation
- final
- precision
- unaryMinus

- **constant** ↑

Set to true if this object represent a stationary variable or parameter.

- **equation** ↑

The property storing the expression as a char.

- **final** ↑

Set to true to make all operators stop working on an equation

- **precision** ↑

The precision on the converted numbers. See the nb_num2str function. Default is [], i.e. use num2str without any input.

- **uniaryMinus** ↑

Indicator for last method call being unary minus

Methods:

- | | | | | |
|----------------------|-----------|-----------|-----------|----------------|
| • bgp | • detrend | • disp | • exp | • log |
| • logncdf | • lognpdf | • minus | • mpower | • mrdivide |
| • mtimes | • normcdf | • norminv | • normpdf | • plus |
| • power | • rdivide | • sgp | • sqrt | • steady_state |
| • steady_state_first | • times | • uminus | • uplus | |

► **bgp** ↑

obj = bgp(obj)

Description:

Balanced growth path operator.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **detrend** ↑

```
obj = detrend(obj)
```

Description:

Detrend operator.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **disp** ↑

```
disp(obj)
```

Description:

Display object (In the command window)

Input:

- obj : An object of class nb_bgrowth

Output:

The object displayed on the command line.

Examples:

```
obj (Note without semicolon)
```

Written by Kenneth S. Paulsen

► **exp** ↑

```
obj = exp(obj)
```

Description:

Exponential function.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► **log** ↑

obj = log(obj)

Description:

Log function (natural logarithm).

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► **logncdf** ↑

obj = logncdf(obj,m,k)

Description:

CDF of the log normal distribution.

Input:

- obj : An object of class nb_bgrowth.

- m : The mean of the associated normal distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

- k : The std of the associated normal distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► lognpdf ↑

```
obj = lognpdf(obj)
```

Description:

PDF of the log normal distribution.

Input:

- obj : An object of class nb_bgrowth.
- m : The mean of the associated normal distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.
- k : The std of the associated normal distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► minus ↑

```
obj = minus(obj,another)
```

Description:

Minus operator (-).

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (^). This will call power (.^)!

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **mrddivide** ↑

```
obj = mrddivide(obj,another)
```

Description:

Right division operator (/), same as (./).

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*). Here assumed equal to .* operator.

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► normcdf ↑

```
obj = normcdf(obj)
```

Description:

CDF of the normal distribution.

Input:

- obj : An object of class nb_bgrowth.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► norminv ↑

```
obj = norminv(obj)
```

Description:

Inverse normal cdf.

Input:

- obj : An object of class nb_bgrowth.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► normpdf ↑

```
obj = normcdf(obj)
```

Description:

PDF of the normal distribution.

Input:

- obj : An object of class nb_bgrowth.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_bgrowth object.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► plus ↑

```
obj = plus(obj,another)
```

Description:

Plus operator (+).

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **power** ↑

```
obj = power(obj,another)
```

Description:

Power operator (.^).

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

```
obj = rdivide(obj,another)
```

Description:

Right division operator (./).

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **sgp** ↑

```
obj = sgp(obj)
```

Description:

Stochastic growth path operator.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► **steady_state** ↑

obj = steady_state(obj)

Description:

Steady-state operator.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► **steady_state_first** ↑

```
obj = steady_state_first(obj)
```

Description:

Steady-state (first regime) operator.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*) .

Input:

- another : A scalar number, nb_bgrowth object or string.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

► uplus ↑

```
obj = uplus(obj)
```

Description:

Unary plus.

Input:

- obj : An object of class nb_bgrowth.

Output:

- obj : An object of class nb_bgrowth.

Written by Kenneth SÃ¶terhagen Paulsen

■ nb_eq2Latex

Go to: [Properties](#) | [Methods](#)

A class for converting equations to latex format.

Constructor:

```
obj = nb_eq2Latex(varName,precision);
```

Input:

- varName : Name of the variable(s) of the expression(s) that is converted to latex code, as a one line char or a cellstr.

- precision : See help on the property precision.

Output:

- obj : An object of class nb_eq2Latex.

Examples:

```
d = nb_eq2Latex('x');
d = d.^2
```

See also:

[nb_eq2Latex.parse](#), [nb_dsge.writeTex](#)

Written by Kenneth Søsterhagen Paulsen

Properties:

- [latex](#)
- [precision](#)
- [values](#)

- **latex** [↑](#)

The property storing the latex code of the equation, as a one line char.

- **precision** [↑](#)

The precision on the converted numbers. See the [nb_num2str](#) function. Default is 14, i.e. use [nb_num2str](#) with the second input set to 14.

- **values** [↑](#)

The property storing the original equation as a string.

Methods:

- | | | | |
|--------------------------------------|----------------------------|--------------------------|--------------------------------|
| • addLatexPar | • eq | • exp | • gap |
| • ge | • gt | • le | • log |
| • logncdf | • lognpdf | • lt | • minus |
| • mpower | • mrdivide | • mtimes | • ne |
| • normcdf | • normpdf | • parse | • plus |
| • power | • rdivide | • sqrt | • steady_state |
| • steady_state_first | • times | • uminus | • uplus |
| • writePDF | • writeTex | | |

► **addLatexPar** [↑](#)

`str = nb_eq2Latex.addLatexPar(str,type)`

Description:

Add latex matched parentheses if needed to a term during parsing.

Input:

- `str` : A one line char.
- `type` : Give true to always add parentheses, even if no `+, -,` or `\frac` is found.

Output:

- str : A one line char.

Written by Kenneth SÃ¶terhagen Paulsen

► eq ↑

```
obj = eq(obj,another,flip)
```

Description:

Equality operator (==), will be subst. by the = sign in the latex equations!

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► exp ↑

```
obj = exp(obj)
```

Description:

Exponential.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **gap** ↑

```
obj = gap(obj)
```

Description:

The gap operator.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **ge** ↑

```
obj = ge(obj,another,flip)
```

Description:

Greater than or equal operator (\geq).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **gt** ↑

```
obj = gt(obj,another,flip)
```

Description:

Greater than operator (>).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- another : A scalar number, nb_eq2Latex object, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **le** ↑

obj = le(obj,another,flip)

Description:

Less than or equal operator (<=)

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **log** ↑

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **logncdf** ↑

```
obj = logncdf(obj,m,k)
```

Description:

Log normal cdf.

Input:

- obj : An object of class nb_eq2Latex.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or an object of class nb_param.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or an object of class nb_param.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **lognpdf** ↑

```
obj = lognpdf(obj,m,k)
```

Description:

Log normal pdf.

Input:

- obj : An object of class nb_eq2Latex.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a one line char representing a number.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a one line char representing a number.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **lt** ↑

obj = lt(obj,another,flip)

Description:

Less than operator (<).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

obj = minus(obj,another,flip)

Description:

Minus operator (-).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (^). This will call power (.^)!

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **mrddivide** ↑

obj = mrddivide(obj,another)

Description:

Right division operator (/), same as (./).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sætherhagen Paulsen

► **mtimes** ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*). Here assumed equal to .* operator.

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sætherhagen Paulsen

► **ne** ↑

```
obj = ne(obj,another,flip)
```

Description:

Not equal to operator (~=).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sætherhagen Paulsen

► **normcdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : An object of class nb_eq2Latex.
- m : The mean of the distribution. Either as a scalar double or a one line char representing a number.
- k : The std of the distribution. Either as a scalar double or a one line char representing a number.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **normpdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : An object of class nb_eq2Latex.
- m : The mean of the distribution. Either as a scalar double or a one line char representing a number.
- k : The std of the distribution. Either as a scalar double or a one line char representing a number.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **parse** ↑

```
obj = nb_eq2Latex.parse(expr,varargin)
```

Description:

Parse mathematical expressions and turn them into a vector of nb_eq2Latex objects.

Input:

- expr : A N x 1 cellstr with the mathematical expressions to parse.

Optional input:

- 'latexVars' : Use this input to translate the names of the variables/parameters of the mathematical expression to latex names. In this case 'vars' must also be given. Each element of the 'vars' input is translated into the corresponding element of this input. Must be a M x 1 cellstr.

- 'precision' : See help on the precision property. Default is 14.

- 'vars' : A M x 1 cellstr with the variables of the expression. If not provided or empty, the variables will be parsed at the cost of speed.

Output:

- obj : A N x 1 vector of nb_eq2Latex objects.

See also:

[nb_eq2Latex.writePDF](#)

Written by Kenneth Sætherhagen Paulsen

► **plus** ↑

```
obj = plus(obj,another,flip)
```

Description:

Plus operator (+).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **power** ↑

obj = power(obj,another,flip)

Description:

Power operator ($.^$).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

obj = rdivide(obj,another,flip)

Description:

Right division operator ($./$).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **steady_state** ↑

obj = steady_state(obj)

Description:

The steady_state operator.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth Sæterhagen Paulsen

► **steady_state_first** ↑

```
obj = steady_state_first(obj)
```

Description:

The steady_state (first regime) operator.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **times** ↑

```
obj = times(obj,another,flip)
```

Description:

Times operator (.*).

Input:

- obj : An object of class nb_eq2Latex, scalar double or string.
- another : An object of class nb_eq2Latex, scalar double or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **uplus** ↑

obj = uplus(obj)

Description:

Uniary plus.

Input:

- obj : An object of class nb_eq2Latex.

Output:

- obj : An object of class nb_eq2Latex.

Written by Kenneth SÃ¶terhagen Paulsen

► **writePDF** ↑

writePDF(obj,filename)

Description:

Write the equations to PDF

Input:

- obj : An object of class nb_eq2Latex.

- filename : The filename of the saved pdf file. As a one line char.
Extension is ignored.

Examples:

obj.writePDF('test')

Written by Kenneth S. Paulsen

► **writeTex** ↑

```
code = writeTex(obj,filename)
```

Description:

Write tex file of the equations.

Input:

- obj : An object of class nb_eq2Latex.
- filename : Give the name of the written .tex file. If not given or empty no file will be written. Extension is ignored.

Output:

- code : A Q x 1 cellstr with the latex code.

Written by Kenneth Sæterhagen Paulsen

■ **nb_equation**

Go to: [Properties](#) | [Methods](#)

A class for representing an equation with a set of nb_term objects.

Superclasses:

nb_term

Constructor:

```
obj = nb_equation(expr);
```

Input:

- expr : Expression to simplify, as a one line char or a cellstr.

Output:

- obj : An object of class nb_equation.

Examples:

```
terms = nb_equation('x+x*y');
```

See also:

[nb_term](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- `numberOfTerms` • `operator` • `terms`
- **`numberOfTerms` ↑**

The number of terms. As a scalar integer.

Inherited from superclass NB_TERM

- **`operator` ↑**

Type of operator splitting the terms.

Inherited from superclass NB_TERM

- **`terms` ↑**

A vector of nb_term objects.

Inherited from superclass NB_TERM

Methods:

- `callExpOnSub` • `callFuncOnSub` • `cellstr` • `clean` • `correct`
- `eq` • `equalToThePower` • `exp` • `gap` • `generalFunc`
- `getTerms` • `initialize` • `intersect` • `isempty` • `ismember`
- `log` • `logncdf` • `lognpdf` • `minus` • `mpower`
- `mrdivide` • `mtimes` • `normcdf` • `norminv` • `normpdf`
- `plus` • `power` • `rdivide` • `removePar` • `simplify`
- `sort` • `split` • `sqrt` • `steady_state` • `times`
- `toString` • `uminus` • `union` • `unique` • `uplus`

► **`callExpOnSub` ↑**

`obj = callExpOnSub(obj)`

See also:

`nb_term.log`

Written by Kenneth Sæterhagen Paulsen

► **`callFuncOnSub` ↑**

`obj = callFuncOnSub(obj, func, varargin)`

See also:

[nb_term.generalFunc](#)

Written by Kenneth Sæterhagen Paulsen

► **cellstr** ↑

c = toString(obj)

Description:

Convert array of nb_term objects to cellstr.

Inputs:

- obj : An array of nb_term objects.

Output:

- c : A cellstr with same size as obj.

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **clean** ↑

obj = clean(obj)

Description:

Clean up nb_term object after operations.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **correct** ↑

expr = nb_term.correct(expr)

Description:

Correct term for $\cdot*$, $\cdot/$ and $\cdot^$ operators.

Input:

- expr : A one line char with a mathematical expression.

Output:

- expr : Corrected version of the input.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **eq** ↑

```
test = eq(obj,another)
```

Description:

Test for equality of two objects. They are only equal if they represents the same equation.

Inputs:

- obj : A nb_equation object.
- another : A nb_equation object.

Output:

- test : true or false.

Written by Kenneth SÃ¶terhagen Paulsen

► **equalToThePower** ↑

```
[test,obj] = equalToThePower(obj,another)
```

Description:

Test for equality of two objects ignoring powers.

Inputs:

- obj : A nb_equation object.
- another : A nb_equation object.

Output:

- test : true or false.
- obj : A nb_equation object representing the simplified object. E.g:
$$\begin{aligned} \text{eq}^*(\text{eq})^{-1} &\rightarrow 1, \quad \text{eq}^2 * \text{eq}^1 \rightarrow \text{eq}^3 \\ \text{eq} / (\text{eq} + \text{eq} * x) &\rightarrow 1 / (1 + x) \end{aligned}$$
But not yet:
$$\text{eq}^2 / (\text{eq}^2 + \text{eq} * x) \rightarrow \text{eq} / (\text{eq} + x)$$

Written by Kenneth SÃ¶terhagen Paulsen

► exp ↑

obj = exp(obj)

Description:

exp operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► gap ↑

obj = gap(obj)

Description:

Gap operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **generalFunc** ↑

```
obj = generalFunc(obj, func, varargin)
```

Description:

Call general function on a nb_term objects.

Input:

- obj : A vector of nb_term objects.
- func : A one line char with the function to apply on the nb_term objects.

Optional input:

- nb_term objects representing the extra inputs to the function.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **getTerms** ↑

```
terms = getTerms(obj)
```

Description:

Get the expression(s) the object(s) represents.

Input:

- obj : A Nx1 vector of nb_term objects.

Output:

- expr : A Nx1 cellstr with the terms of the nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **initialize** ↑

```
obj = nb_equation.initialize(rows,cols)
```

Description:

Initialize vector of nb_equation objects.

Input:

- rows : Number of rows. As a scalar integer.
- cols : Number of columns. As a scalar integer.

Output:

- obj : A rows x cols nb_term object.

Written by Kenneth Sætherhagen Paulsen

► **intersect ↑**

```
int = intersect(obj,another)
```

Description:

Get the intersect of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- int : The intersect, as a vector of nb_term objects.

Written by Kenneth Sætherhagen Paulsen

Inherited from superclass NB_TERM

► **isempty ↑**

```
ret = isempty(obj)
```

Description:

Is the nb_term object empty or not. Also work for matrices of nb_term objects.

Input:

- obj : A nb_term object (matrix)

Output:

- ret : A logical with same size as obj.

Written by Kenneth SÃ¶terhagen Paulsen

► **ismember** ↑

```
[ind,loc] = ismember(obj,another)
```

Description:

Check if the terms in obj is member of the terms in another.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- ind : A logical vector with same size as obj. Elements that are true are found to be in the another input.
- loc : A double vector with same size as obj. Locations of the elements that are found in the another input. If not found 0 is returned.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **log** ↑

```
obj = log(obj)
```

Description:

log operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► logncdf ↑

```
obj = logncdf(obj,m,k)
```

Description:

Log-normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► lognpdf ↑

```
obj = lognpdf(obj,m,k)
```

Description:

Log-normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **minus** ↑

obj = minus(obj,another)

Description:

Minus operator (-) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mrdivide** ↑

obj = mrdivide(obj,another)

Description:

Division operator (/) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mtimes** ↑

obj = mtimes(obj,another)

Description:

```
Times operator (*) for nb_term objects.
```

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **normcdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **norminv** ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **normpdf** ↑

obj = normpdf(obj,m,k)

Description:

Normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **plus** ↑

obj = plus(obj,another)

Description:

Plus operator (+) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **power** ↑

obj = power(obj,another)

Description:

Power operator (.^) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **rdivide** ↑

```
obj = rdivide(obj,another)
```

Description:

Division operator (./) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **removePar** ↑

```
expr = removePar(expr)
```

Description:

Is the expression enclosed with parentheses?

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **simplify** ↑

```
expr = nb_term.simplify(expr)
expr = nb_term.simplify(expr,waitbar)
```

Description:

Simplify the mathematical expressions.

Input:

- expr : A one line char with a mathematical expression or a cellstr with the mathematical expressions.
- waitbar : A nb_waitbar5 object. It will use the next available waitbar of the given object. If empty no waitbar is added.

Output:

- obj : A cellstr with the simplified mathematical expressions.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **sort ↑**

```
sorted = sort(obj)
```

Description:

Sort terms in an equation.

Written by Kenneth SÃ¶terhagen Paulsen

► **split ↑**

```
obj = nb_term.split(expr)
obj = nb_term.split(expr,waitbar)
```

Description:

Split the mathematical expression into separate terms.

Input:

- expr : A one line char with a mathematical expression.
- waitbar : A nb_waitbar5 object. If empty no waitbar is added.

Output:

- obj : A nb_term object representing the mathematical expression.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

sqrt operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **steady_state** ↑

```
obj = steady_state(obj)
```

Description:

Steady-state operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **times** ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► [toString ↑](#)

c = toString(obj)

Description:

Convert nb_num object to string or cellstr.

Inputs:

- obj : A nb_num object. May also be a matrix.

Output:

- c : If obj is scalar the output will be a one line char, otherwise it will be a cellstr with same size as obj.

Written by Kenneth Sæterhagen Paulsen

► [uminus ↑](#)

obj = uminus(obj)

Description:

Unary minus operator (-) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **union** ↑

uni = union(obj,another)

Description:

Get the union of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.

- another : A vector of nb_term objects.

Output:

- uni : The union, as a vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **unique** ↑

uniq = unique(obj,another)

Description:

Locate the unique nb_term objects in a vector of nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- uniq : The unique nb_term objects.

- ind : Vector such that uniq = obj(ind).

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► uplus ↑

```
obj = uplus(obj)
```

Description:

Unary plus opertor (+) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

■ nb_logLinearize

Go to: [Properties](#) | [Methods](#)

A class for doing symbolic log-linearization of a mathematical expression.

Constructor:

```
obj = nb_logLinearize(equations,variables,parameters,precision);
```

Input:

- equations : Name of the variable to take derivative respect to.

- variables : The variables to do the log-linearization wrt. I.e. the rest are treated as parameters.

- parameters : You can hard code in the parameters, if you already know them, otherwise these will be parsed out. If given it must be a 1 x nParam cellstr.

- precision : See the doc on the property precision.

Output:

- obj : An object of class nb_logLinearize.

Examples:

```
logLin = nb_logLinearize({'x+y','z=x*y^alpha'},{'x','y','z'});  
logLin = logLinearize(logLin);  
logLinEq = getLinearization(logLin);
```

See also:

[nb_mySD](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- deriv
- doSimplify
- eqFunc
- equations
- gapIdentifier
- logLinEq
- parameters
- precision
- ssEq
- ssIdentifier
- variables

- **deriv** ↑

A nb_mySD object storing the symbolic derivatives.

- **doSimplify** ↑

Set to false to prevent using nb_term.split to simplify the log-linearized equations. Default is false.

- **eqFunc** ↑

Function handle representing the equations.

- **equations** ↑

The equation we start with. As a N x 1 cellstr. Include a equality sign ('=') to indicate that is a equation and not only a term to log linearize.

- **gapIdentifier** ↑

The text that is used to enclose gap variables. Default is 'x_GAP'. Must be a 1x2 cellstr. The first element is added before the variable, and the second after.

- **logLinEq** ↑

The log-linearized equation. As a N x 1 cellstr.

- **parameters** ↑

A cellstr with the parameters. If not provided, they will be parsed out automatically at a cost of speed!

- **precision** ↑

The precision on the converted numbers. See the nb_num2str function. Default is 14, i.e. use nb_num2str with the second input set to 14.

- **ssEq** ↑

Equations in steady-state. As a N x 1 cellstr.

- **ssIdentifier** ↑

The text that is used to enclose steady-state variables. Default is 'steady_state(x)'. Must be a 1x2 cellstr. The first element is added before the variable, and the second after.

- **variables** ↑

A cellstr with the variables to log-linearize wrt.

Methods:

- [cellstr](#)
 - [logLinearize](#)
-

► **cellstr** ↑

eqs = cellstr(obj)

Description:

Get the equations as a cellstr.

Input:

- obj : An object of class nb_logLinearize
- type : Either 'logLinEq' (log-linearized equations, default), 'equations' (original equations) or 'ssEq' (equations in steady-state)

Output:

- eqs : A cellstr.

Written by Kenneth Sæterhagen Paulsen

► **logLinearize** ↑

obj = logLinearize(obj)

Description:

Do the log linearization.

Input:

- obj : An object of class nb_logLinearize

Output:

- obj : An object of class nb_logLinearize. See the property logLinEq for the log-linearized equations.

Written by Kenneth Sæterhagen Paulsen

■ nb_logger

Go to: [Properties](#) | [Methods](#)

A class for logging.

Superclasses:

handle

Constructor:

logger = nb_logger(folder,level)

Input:

- folder : Specify the folder to log to. Will set the environment variable 'LOGGERFILE'.

- level : Level of logging. Default is 0, i.e. to log everything. Will set the environment variable 'LOGGERLEVEL'.

Output:

- logger : A nb_logger object.

Written by Kenneth Sæterhagen Paulsen

Properties:

- ALL
- DEBUG
- DEFAULT
- ERROR
- INFO
- OFF
- WARN

- ALL ↑

nb_logger.ALL is a property.

- DEBUG ↑

nb_logger.DEBUG is a property.

- DEFAULT ↑

`nb_logger.DEFAULT` is a property.

- **ERROR** ↑

`nb_logger.ERROR` is a property.

- **INFO** ↑

`nb_logger.INFO` is a property.

- **OFF** ↑

`nb_logger.OFF` is a property.

- **WARN** ↑

`nb_logger.WARN` is a property.

Methods:

- `closeLoggerFile`
 - `getLoggerFolder`
 - `getParallelLoggerFile`
 - `logging`
 - `loggingDuringParallel`
 - `openLogFile`
-

► **closeLoggerFile** ↑

`nb_logger.closeLoggerFile(inputs)`

Description:

Close logger file.

Written by Kenneth Sæterhagen Paulsen

► **getLoggerFolder** ↑

`folder = nb_logger.getLoggerFolder()`

Description:

Get logger folder.

Written by Kenneth Sæterhagen Paulsen

► **getParallelLoggerFile** ↑

```
w = nb_logger.getParallelLoggerFile(type)
```

Description:

Write logger file.

Written by Kenneth Sæterhagen Paulsen

► **logging** ↑

```
nb_logger.logging(messageLevel,inputs,message,Err)
```

Description:

Write logger file.

Written by Kenneth Sæterhagen Paulsen

► **loggingDuringParallel** ↑

```
nb_logger.loggingDuringParallel(messageLevel,write,w,message,Err)
```

Description:

Write logger file.

Written by Kenneth Sæterhagen Paulsen

► **openLoggerFile** ↑

```
inputs = nb_logger.openLoggerFile(inputs,obj)
```

Description:

Open logger file.

Written by Kenneth Sæterhagen Paulsen

■ **nb_macro**

Go to: [Properties](#) | [Methods](#)

A class representing a macro variable that can be defined in a NB toolbox macro syntax.

Constructor:

```
obj = nb_macroVar(input)
```

Input:

- input : One of:
 - > Scalar double.
 - > Vector of numerical.
 - > Matrix of numerical, will be converted to a vector.
 - > One line char. Special cases;
 - * '"C"' will be converted to 'C'.
 - * '["C","B"]' will be converted to cellstr.
 - * '{"C","B"}' will be converted to cellstr.
 - * {'C','B'}' will be converted to cellstr.
 - * '2' will be converted to scalar numerical.
 - * 'true' or 'false' will be converted to scalar logical.
 - * '[1,0.2,3]' will be converted to a numerical vector.
 - * '[true,false]' will be converted to a logical vector.
 - * '[true,false,1]' will be converted to a numerical vector.
 - * Else it will be stored as a normal char.
 - > Multi-line char, converted to cellstr. No interpretation!
 - > Scalar logical.
 - > Vector of logicals.
 - > Matrix of logicals, will be converted to a vector.

Output:

- obj : A object of class nb_macroVar.

Written by Kenneth Sætherhagen Paulsen

Properties:

- name • value

- **name** ↑

Name of the the macro variable

- **value** ↑

Value that the macro variable represents

Methods:

• and	• colon	• define	• disp	• eq
• eval	• forEnd	• ge	• gt	• ifElseEnd
• in	• indexing	• interpret	• le	• lt
• minus	• mpower	• mrdivide	• mtimes	• ne
• not	• or	• parse	• plus	• power
• rdivide	• times	• type		

► **and** ↑

```
obj = and(obj1,obj2)
```

Description:

The && operator for nb_macro objects. This allows for;
> logical = numeric && numeric
> logical = logical && logical
> logical = logical && numeric

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) && 4 will result in a nb_macro object representing the logical true.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

or, not

Written by Kenneth Sæterhagen Paulsen

► colon ↑

```
obj = colon(obj1,obj2)
```

Description:

The : operator for nb_macro objects. This allows for;
> numeric = numeric : numeric

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',1) : 2 will result in a nb_macro object representing the double vector [1,2].

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

Written by Kenneth Sæterhagen Paulsen

► **define** ↑

`obj = define(obj, name, statement)`

Description:

Define a new macro variable.

Input:

- obj : A 1 x N vector of nb_macro objects.
- name : A one line char with the name of the defined variable.
- statement : A one line char to interpret.

Output:

- obj : A 1 x (N + 1) vector of nb_macro objects.

See also:

[nb_macro.parse](#)

Written by Kenneth Sæterhagen Paulsen

► **disp** ↑

`disp(obj)`

Description:

Displaying the macro variable in the command window.

Input:

- obj : A nb_macro object of any size.

Written by Kenneth Sæterhagen Paulsen

► **eq** ↑

obj = eq(obj1,obj2)

Description:

The == operator for nb_macro objects. This allows for;

```
> logical = numeric == numeric
> logical = logical == logical
> logical = logical == numeric
> logical = char == char = strcmp(char,char)
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) == 4 will result in a nb_macro object representing the logical false.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.

- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[lt](#), [gt](#), [le](#), [ge](#), [ne](#)

Written by Kenneth Sæterhagen Paulsen

► **eval** ↑

value = eval(obj,expression)

Description:

Evaluate a expression using the macro processor language.

Input:

- obj : A vector of nb_macro objects.
- expression : A one line char.

Output:

- value : A scalar nb_macro object.

See also:

[nb_eval](#)

Written by Kenneth Sæterhagen Paulsen

► **forEnd** ↑

[obj,statement] = forEnd(obj,forVar,looped,forStatements)

Description:

Running an @#for @#endfor block in the macro processing language.

Input:

- obj : A vector of nb_macro objects storing the macro variables.
- forVar : A one line char with the name of the for loop variable.
- looped : Either a one line char or a scalar nb_macro object.
- forStatements : The statements in the body of the @#for loop, as a N x cellstr.

Output:

- statement : A Q x 1 cellstr with the parsed version of the model file statements.

See also:

[nb_macro.ifElseEnd](#), [nb_macro.parse](#)

Written by Kenneth Sæterhagen Paulsen

► **ge** ↑

```
obj = ge(obj1,obj2)
```

Description:

The `>=` operator for nb_macro objects. This allows for;

```
> logical = numeric >= numeric  
> logical = logical >= logical  
> logical = logical >= numeric
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. `nb_macro('a',2) >= 4` will result in a nb_macro object representing the logical false.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- `obj1` : An object of any class.
- `obj2` : An object of any class.

Output:

- `obj` : An object of class nb_macro.

See also:

[ne](#), [lt](#), [le](#), [gt](#), [eq](#)

Written by Kenneth Sæterhagen Paulsen

► **gt** ↑

```
obj = gt(obj1,obj2)
```

Description:

The `>` operator for nb_macro objects. This allows for;

```
> logical = numeric > numeric  
> logical = logical > logical  
> logical = logical > numeric
```

Caution : The above type is refer to as the type property of the

nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) > 4 will result in a nb_macro object representing the logical false.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[ne](#), [lt](#), [le](#), [ge](#), [eq](#)

Written by Kenneth Sæterhagen Paulsen

► **ifElseEnd ↑**

[obj,statement] = ifElseEnd(obj,condition,statementTrue,statementFalse)

Description:

Running an @#if @#else @#endif block in the macro processing language.

Input:

- obj : A vector of nb_macro objects storing the macro variables.
- condition : The condition to be tested. If true the statementTrue lines will be used.
- statementTrue : A N x 1 (3) cellstr with a set of model file statements.
- statementFalse : A M x 1 (3) cellstr with a set of model file statements.

Output:

- statement : A Q x 1 cellstr with the parsed version of the model file statements.

See also:

[nb_macro.forEnd](#), [nb_macro.parse](#)

Written by Kenneth Sæterhagen Paulsen

► [in](#) ↑

`obj = in(obj,objIn)`

Description:

Check if the value represented by obj can be found in objIn.

Caution: If you try to check if a number is part of a cellstr array, or a char is part of a numerical (or logical) it will not give an error. Instead a nb_macro object representing false will be returned.

Input:

- `obj` : A nb_macro object.
- `objIn` : A nb_macro object.

Output:

- `obj` : A nb_macro object representing true or false.

Examples:

```
mString = nb_macro('var1','C');
mString2 = nb_macro('var2','ABC');
mLogical = mString.in(mString2)

mString = nb_macro('var1','C');
mArray = nb_macro('array1',{'C','D','E'});
mLogical = mString.in(mArray)

mNum = nb_macro('var1',2);
mArray = nb_macro('array1',[1,3,5]);
mLogical = mNum.in(mArray)

mNum = nb_macro('var1',2);
mNum2 = nb_macro('var2',1);
mLogical = mNum.in(mNum2)
```

Written by Kenneth Sæterhagen Paulsen

► [indexing](#) ↑

```
obj = indexing(obj, index)
```

Description:

Use this method to index the arrays that the nb_macro object represent.

Input:

- obj : An object of nb_macro.
- index : A numerical array of integers. E.g. 5, 1:4, [1,3], [1,3:5]

Output:

- obj : An object of nb_macro.

Examples:

```
m = nb_macro('I','DE');
m = indexing(m,2)

m = nb_macro('I2',{ 'A','B','C','D' });
m = indexing(m,1:3)
```

Written by Kenneth SÃ¶terhagen Paulsen

► **interpret** ↑

```
obj = nb_macro.interpret(input)
```

Description:

Convert input to a vector of nb_macro objects.

Input:

- input : Either a 1 x N or N x 1 nb_macro object, struct with N fields or a N x 2 cell.

Output:

- obj : An object of class nb_macro with size 1 x N.

See also:

[nb_dsge](#)

Written by Kenneth SÃ¶terhagen Paulsen

► **le** ↑

```
obj = le(obj1,obj2)
```

Description:

The <= operator for nb_macro objects. This allows for;

```
> logical = numeric <= numeric  
> logical = logical <= logical  
> logical = logical <= numeric
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) <= 4 will result in a nb_macro object representing the logical true.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[ne](#), [gt](#), [lt](#), [ge](#), [eq](#)

Written by Kenneth Sæterhagen Paulsen

► **lt** ↑

```
obj = lt(obj1,obj2)
```

Description:

The < operator for nb_macro objects. This allows for;

```
> logical = numeric < numeric  
> logical = logical < logical  
> logical = logical < numeric
```

Caution : The above type is refer to as the type property of the

nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) <= 4 will result in a nb_macro object representing the logical true.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

ne, gt, le, ge, eq

Written by Kenneth Sæterhagen Paulsen

► **minus ↑**

obj = minus(obj1,obj2)

Description:

The - operator for nb_macro objects. This allows for;

```
> numeric = numeric - numeric
> numeric = logical - logical
> numeric = logical - numeric
> cellstr = cellstr - cellstr = setdiff(cellstr, cellstr)
> char = char - char = setdiff(char,char)
> cellstr = cellstr - char = setdiff(cellstr,{char})
> cellstr = char - cellstr = setdiff({char},cellstr)
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',{'C','D'}) - {'C'} will result in a nb_macro object representing the cellstr {'D'}.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[plus](#), [times](#), [power](#), [rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

obj = mpower(obj1,obj2)

Description:

Redirect to power, i.e. .^ and ^ is the same operator.

See also:

[power](#)

Written by Kenneth Sæterhagen Paulsen

► **mrdivide** ↑

obj = mrdivide(obj1,obj2)

Description:

Redirect to times, i.e. ./ and / is the same operator.

See also:

[rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

```
obj = mtimes(obj1,obj2)
```

Description:

Redirect to times, i.e. .* and * is the same operator.

See also:

[times](#)

Written by Kenneth Sæterhagen Paulsen

► **ne** ↑

```
obj = ne(obj1,obj2)
```

Description:

The ~= operator for nb_macro objects. This allows for;

```
> logical = numeric ~= numeric
> logical = logical ~= logical
> logical = logical ~= numeric
> logical = char ~= char = ~strcmp(char,char)
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) ~= 4 will result in a nb_macro object representing the logical true.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[lt](#), [gt](#), [le](#), [ge](#), [eq](#)

Written by Kenneth Sæterhagen Paulsen

► **not** ↑

```
obj = not(obj)
```

Description:

The ~ operator for nb_macro objects. This allows for;
> logical = ~numeric
> logical = ~logical

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. ~nb_macro('a',2) will result in a nb_macro object representing the logical false.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[ne](#), [lt](#), [le](#), [gt](#), [ge](#), [eq](#)

Written by Kenneth Sæterhagen Paulsen

► **or** ↑

```
obj = or(obj1,obj2)
```

Description:

The || operator for nb_macro objects. This allows for;
> logical = numeric || numeric
> logical = logical || logical
> logical = logical || numeric

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) || 4 will result in a nb_macro object representing the logical true.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

and, not

Written by Kenneth Sæterhagen Paulsen

► **parse ↑**

[obj,parsed] = parse(obj,statements)

Description:

Parse model file that utilizes the macro processing language.

Input:

- obj : A vector of nb_macro objects.
- statements : A N x 3 cell with the model file statements at the first column, the line number at the second and the filenames at the third column.

Caution: This input can also be a N x 1 cell, but then the error messages will be less informative.

Output:

- parsed : A Q x 3 (1) cellstr with the statements input cleeaned for the macro processing language.

See also:

► **plus ↑**

```
obj = plus(obj1,obj2)
```

Description:

The + operator for nb_macro objects. This allows for;

```
> numeric = numeric + numeric
> numeric = logical + logical
> numeric = logical + numeric
> cellstr = cellstr + cellstr = [cellstr, cellstr]
> char    = char + char = [char,char]
> cellstr = cellstr + char = [cellstr,char]
> cellstr = char + cellstr = [char,cellstr]
```

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',{'C','D'}) + {'F'} will result in a nb_macro object representing the cellstr {'C','D','F'}.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

minus, times, power, rdivide

► **power ↑**

```
obj = power(obj1,obj2)
```

Description:

The * operator for nb_macro objects. This allows for;
> numeric = numeric .^ numeric
> numeric = logical .^ logical
> numeric = logical .^ numeric

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the MATLAB basic type. E.g. nb_macro('a',2) .^ 4 will result in a nb_macro object representing the double 16.

Caution : This operator supports nb_macro object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- obj1 : An object of any class.
- obj2 : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[plus](#), [minus](#), [times](#), [rdivide](#)

Written by Kenneth SÅterhagen Paulsen

► **rdivide** ↑

```
obj = rdivide(obj1,obj2)
```

Description:

The * operator for nb_macro objects. This allows for;
> numeric = numeric ./ numeric
> numeric = logical ./ logical
> numeric = logical ./ numeric

Caution : The above type is refer to as the type property of the nb_macro variable.

Caution : This operator is also supported for if one nb_macro object represent one of the above type but the other input is of the

MATLAB basic type. E.g. `nb_macro('a',2) ./ 4` will result in a `nb_macro` object representing the double 0.5.

Caution : This operator supports `nb_macro` object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- `obj1` : An object of any class.
- `obj2` : An object of any class.

Output:

- `obj` : An object of class `nb_macro`.

See also:

[plus](#), [minus](#), [times](#), [power](#)

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

`obj = times(obj1,obj2)`

Description:

The `*` operator for `nb_macro` objects. This allows for;
> numeric = numeric `.*` numeric
> numeric = logical `.*` logical
> numeric = logical `.*` numeric

Caution : The above type is refer to as the type property of the `nb_macro` variable.

Caution : This operator is also supported for if one `nb_macro` object represent one of the above type but the other input is of the MATLAB basic type. E.g. `nb_macro('a',2) .* 4` will result in a `nb_macro` object representing the double 8.

Caution : This operator supports `nb_macro` object, numeric and logical matrices, as long as one is a scalar or their sizes matches.

Input:

- `obj1` : An object of any class.
- `obj2` : An object of any class.

Output:

- obj : An object of class nb_macro.

See also:

[plus](#), [minus](#), [power](#), [rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **type** ↑

t = type(obj)

Description:

Get the MATLAB basic class the nb_macro object represent.

Input:

- obj : An object of class nb_macro.

Output:

- t : If numel(obj) > 0: cellstr
else : one line char

Written by Kenneth Sæterhagen Paulsen

■ **nb_mySD**

Go to: [Properties](#) | [Methods](#)

A class for doing symbolic derivative of a function.

Superclasses:

[nb_varOrPar](#), [matlab.mixin.Heterogeneous](#)

Constructor:

obj = nb_mySD(varName,precision);

Input:

- varName : Name of the variable to take derivative respect to, as a one line char or a cellstr.

- precision : See help on the property precision.

Output:

- obj : An object of class nb_mySD

Examples:

```
d      = nb_mySD('x');
d      = d.^2;
strDer = d.derivatives;
```

See also:

[myAD](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

- [bases](#)
- [derivatives](#)
- [precision](#)
- [values](#)

- **bases** [↑](#)

The base variable.

- **derivatives** [↑](#)

The property storing the derivatives as a cellstr.

- **precision** [↑](#)

The precision on the converted numbers. See the `nb_num2str` function. Default is 14, i.e. use `nb_num2str` with the second input set to 14.

- **values** [↑](#)

The property storing the original equation as a string.

Methods:

- | | | | | |
|--------------------------|---------------------------|---------------------------|--------------------------------|--------------------------------------|
| • addPar | • exp | • log | • logncdf | • lognpdf |
| • max | • min | • minus | • mpower | • mrdivide |
| • mtimes | • normcdf | • norminv | • normpdf | • plus |
| • power | • rdivide | • sqrt | • steady_state | • steady_state_first |
| • times | • uminus | • uplus | | |

► **addPar** [↑](#)

```
str = nb_mySD.addPar(str,type)
```

Description:

Add parentheses if needed to a term during parsing.

Input:

- str : A one line char.
- type : Give false to allways add parentheses, even if no +,-,
^ or * is found.

Output:

- str : A one line char.

Written by Kenneth SÃ¶terhagen Paulsen

► **exp ↑**

obj = exp(obj)

Description:

Exponential.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth SÃ¶terhagen Paulsen

► **log ↑**

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth SÃ¶terhagen Paulsen

► **logncdf** ↑

```
obj = logncdf(obj,m,k)
```

Description:

Log normal cdf.

Input:

- obj : An object of class nb_mySD.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or a nb_param object.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or a nb_param object.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **lognpdf** ↑

```
obj = lognpdf(obj,m,k)
```

Description:

Log normal pdf.

Input:

- obj : An object of class nb_mySD.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or a nb_param object.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double, a one line char representing a number or a nb_param object.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **max** ↑

```
obj = max(obj,another,flip)
```

Description:

Max operator.

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **min** ↑

```
obj = min(obj,another,flip)
```

Description:

Min operator.

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

```
obj = minus(obj,another,flip)
```

Description:

Minus operator (-).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (^). This will call power (.^)!

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mrddivide** ↑

```
obj = mrddivide(obj,another)
```

Description:

Right division operator (/), same as (./).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

obj = mtimes(obj,another)

Description:

Times operator (*). Here assumed equal to .* operator.

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **normcdf** ↑

obj = normcdf(obj,m,k)

Description:

Normal cdf.

Input:

- obj : An object of class nb_mySD.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **norminv** ↑

obj = norminv(obj,m,k)

Description:

Inverse normal cdf.

Input:

- obj : An object of class nb_mySD.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **normpdf** ↑

obj = normpdf(obj,m,k)

Description:

Normal pdf.

Input:

- obj : An object of class nb_mySD.
- m : The mean of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.
- k : The std of the distribution. Either as a scalar double, a one line char representing a number or a nb_param object.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **plus ↑**

obj = plus(obj,another,flip)

Description:

Plus operator (+).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **power ↑**

obj = power(obj,another,flip)

Description:

Power operator (.^).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sætherhagen Paulsen

► **rdivide** ↑

obj = rdivide(obj,another,flip)

Description:

Right division operator (./).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sætherhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **steady_state** ↑

obj = steady_state(obj)

Description:

Derivative of the steady_state operator. Used by nb_DSGE.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_param.

Written by Kenneth Sæterhagen Paulsen

► **steady_state_first** ↑

obj = steady_state_first(obj)

Description:

Derivative of the steady_state_first operator. Used by nb_DSGE.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_param.

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

obj = times(obj,another,flip)

Description:

Times operator (.*).

Input:

- obj : A scalar number, nb_param object, nb_mySD object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.
- flip : Flip the obj and another order.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **uplus** ↑

obj = uplus(obj)

Description:

Unary plus.

Input:

- obj : An object of class nb_mySD.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sætherhagen Paulsen

■ nb_num

Go to: [Properties](#) | [Methods](#)

A class for representing a number. Used by the nb_simplifyEq class.

Superclasses:

nb_term

Constructor:

obj = nb_num(value);

Input:

- value : A scalar double. E.g. 1.

Output:

- obj : An object of class nb_num.

Examples:

base = nb_num(1);

See also:

[nb_term](#)

Written by Kenneth Sætherhagen Paulsen

Properties:

• [numberOfTerms](#) • [operator](#) • [terms](#) • [value](#)

- **numberOfTerms** ↑

The number of terms. As a scalar integer.

Inherited from superclass NB_TERM

- **operator** ↑

Type of operator splitting the terms.

Inherited from superclass NB_TERM

- **terms** ↑

A vector of nb_term objects.

Inherited from superclass NB_TERM

- **value** ↑

The number as a scalar double.

Methods:

- callExpOnSub
- callFuncOnSub
- cellstr
- clean
- correct
- eq
- exp
- gap
- ge
- generalFunc
- gt
- initialize
- intersect
- ismember
- le
- log
- logncdf
- lognpdf
- lt
- minus
- mpower
- mrdivide
- mtimes
- ne
- normcdf
- norminv
- normpdf
- plus
- power
- rdivide
- removePar
- simplify
- split
- sqrt
- steady_state
- times
- toString
- uminus
- union
- unique
- uplus

► **callExpOnSub** ↑

obj = callExpOnSub(obj)

See also:

[nb_term.exp](#)

Written by Kenneth Sæterhagen Paulsen

► **callFuncOnSub** ↑

obj = callFuncOnSub(obj, func, varargin)

See also:

[nb_term.generalFunc](#)

Written by Kenneth Sæterhagen Paulsen

► **cellstr** ↑

```
c = toString(obj)
```

Description:

Convert array of nb_term objects to cellstr.

Inputs:

- obj : An array of nb_term objects.

Output:

- c : A cellstr with same size as obj.

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► [clean](#) ↑

```
obj = clean(obj)
```

Description:

Clean up nb_term object after operations.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► [correct](#) ↑

```
expr = nb_term.correct(expr)
```

Description:

Correct term for .*, ./ and .^ operators.

Input:

- expr : A one line char with a mathematical expression.

Output:

- expr : Corrected version of the input.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **eq** ↑

test = eq(obj,another)

Description:

Test for equality of two objects. They are only equal if the value property is equal.

Inputs:

- obj : A nb_num object or a number.
- another : A nb_num object or a number.

Output:

- test : true or false.

Written by Kenneth Sæterhagen Paulsen

► **exp** ↑

obj = exp(obj)

Description:

exp operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **gap** ↑

```
obj = gap(obj)
```

Description:

Gap operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **ge** ↑

```
test = ge(obj,another)
```

Description:

Test if one object is greater than or equal to another.

Inputs:

- obj : A nb_num object or a number.
- another : A nb_num object or a number.

Output:

- test : true or false.

Written by Kenneth SÃ¶terhagen Paulsen

► **generalFunc** ↑

```
obj = generalFunc(obj,func,varargin)
```

Description:

Call general function on a nb_term objects.

Input:

- obj : A vector of nb_term objects.
- func : A one line char with the function to apply on the nb_term objects.

Optional input:

- nb_term objects representing the extra inputs to the function.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **gt** ↑

test = gt(obj,another)

Description:

Test if one object is greater than another.

Inputs:

- obj : A nb_num object or a number.
- another : A nb_num object or a number.

Output:

- test : true or false.

Written by Kenneth SÃ¶terhagen Paulsen

► **initialize** ↑

obj = nb_term.initialize(rows,cols)

Description:

Initialize vector of nb_term objects.

Input:

- rows : Number of rows. As a scalar integer.
- cols : Number of columns. As a scalar integer.

Output:

- obj : A rows x cols nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **intersect** ↑

```
int = intersect(obj,another)
```

Description:

Get the intersect of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.

- another : A vector of nb_term objects.

Output:

- int : The intersect, as a vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **ismember** ↑

```
[ind,loc] = ismember(obj,another)
```

Description:

Check if the terms in obj is member of the terms in another.

Input:

- obj : A vector of nb_term objects.

- another : A vector of nb_term objects.

Output:

- ind : A logical vector with same size as obj. Elements that are true are found to be in the another input.

- loc : A double vector with same size as obj. Locations of the

elements that are found in the another input. If not found 0 is returned.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **le** ↑

test = le(obj,another)

Description:

Test if one object is less than or equal to another.

Inputs:

- obj : A nb_num object or a number.
- another : A nb_num object or a number.

Output:

- test : true or false.

Written by Kenneth Sæterhagen Paulsen

► **log** ↑

obj = log(obj)

Description:

log operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **logncdf** ↑

```
obj = logncdf(obj,m,k)
```

Description:

Log-normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **lognpdf ↑**

```
obj = lognpdf(obj,m,k)
```

Description:

Log-normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **lt ↑**

```
test = lt(obj,another)
```

Description:

Test if one object is less than another.

Inputs:

- obj : A nb_num object or a scalar number.
- another : A nb_num object or a scalar number.

Output:

- test : true or false.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

obj = minus(obj,another)

Description:

Minus operator (-) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **mrddivide** ↑

obj = mrddivide(obj,another)

Description:

Division operator (/) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_TERM

► **mtimes** ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **ne** ↑

```
test = ne(obj,another)
```

Description:

Test for not equality (~=) of two objects. They are only not equal if the value property is not equal.

Inputs:

- obj : A nb_num object or a number.

- another : A nb_num object or a number.

Output:

- test : true or false.

Written by Kenneth Sæterhagen Paulsen

► **normcdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **norminv** ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **normpdf** ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► plus ↑

```
obj = plus(obj,another)
```

Description:

Plus operator (+) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **power** ↑

```
obj = power(obj,another)
```

Description:

Power operator ($.^$) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **rdivide** ↑

```
obj = rdivide(obj,another)
```

Description:

Division operator ($./$) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► removePar ↑

```
expr = removePar(expr)
```

Description:

Is the expression enclosed with parentheses?

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► simplify ↑

```
expr = nb_term.simplify(expr)
expr = nb_term.simplify(expr,waitbar)
```

Description:

Simplify the mathematical expressions.

Input:

- expr : A one line char with a mathematical expression or a cellstr with the mathematical expressions.
- waitbar : A nb_waitbar5 object. It will use the next available waitbar of the given object. If empty no waitbar is added.

Output:

- obj : A cellstr with the simplified mathematical expressions.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► split ↑

```
obj = nb_term.split(expr)
obj = nb_term.split(expr,waitbar)
```

Description:

Split the mathematical expression into separate terms.

Input:

- expr : A one line char with a mathematical expression.
- waitbar : A nb_waitbar5 object. If empty no waitbar is added.

Output:

- obj : A nb_term object representing the mathematical expression.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **sqrt** ↑

obj = sqrt(obj)

Description:

sqrt operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **steady_state** ↑

obj = steady_state(obj)

Description:

Steady-state operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► times ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► toString ↑

```
c = toString(obj)
```

Description:

Convert nb_num object to string or cellstr.

Inputs:

- obj : A nb_num object. May also be a matrix.

Output:

```
- c : If obj is scalar the output will be a one line char,  
      otherwise it will be a cellstr with same size as obj.
```

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus operator (-) for nb_term objects.

Input:

```
- obj : A vector of nb_term objects.
```

Output:

```
- obj : A vector of nb_term object.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **union** ↑

```
uni = union(obj,another)
```

Description:

Get the union of two vectors of nb_term objects.

Input:

```
- obj      : A vector of nb_term objects.  
- another : A vector of nb_term objects.
```

Output:

```
- uni      : The union, as a vector of nb_term objects.
```

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **unique** ↑

```
uniq = unique(obj,another)
```

Description:

Locate the unique nb_term objects in a vector of nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- uniq : The unique nb_term objects.
- ind : Vector such that uniq = obj(ind).

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus opertor (+) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_TERM

■ **nb_param**

Go to: [Properties](#) | [Methods](#)

A class for storing a parameter. Can be used to calculate symbolic derivatives of expressions with unassign parameters when combined with nb_mySD.

Superclasses:

`nb_varOrPar, matlab.mixin.Heterogeneous`

Constructor:

```
obj = nb_param(parameters);
```

Input:

- parameters : The parameter as a one line char or the parameters as a cellstr.
- precision : See help on the property precision.

Output:

- obj : An object of class nb_param

Examples:

```
p = nb_param({'x','y','z'});  
expr = p(1)*p(2)
```

Written by Kenneth Sæterhagen Paulsen

Properties:

- derivatives • parameter • precision

- **derivatives** ↑

The property storing the derivatives as a cellstr. Always {'0'}.

- **parameter** ↑

A one line char with the parameter or a expression involving the parameter (and possibly others as well)

- **precision** ↑

The precision on the converted numbers. See the nb_num2str function. Default is 14, i.e. use nb_num2str with the second input set to 14.

Methods:

- exp • log • max • min • minus • mpower
- mrdivide • mtimes • plus • power • rdivide • sqrt
- times • uminus • uplus

► **exp** ↑

```
obj = exp(obj)
```

Description:

Exponential.

Input:

- obj : An object of class nb_param.

Output:

- obj : An object of class nb_param.

Written by Kenneth SÃ¶terhagen Paulsen

► **log ↑**

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : An object of class nb_param.

Output:

- obj : An object of class nb_param.

Written by Kenneth SÃ¶terhagen Paulsen

► **max ↑**

obj = max(obj,another)

Description:

Max operator.

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **min** ↑

```
obj = min(obj,another)
```

Description:

Min operator.

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

```
obj = minus(obj,another)
```

Description:

Minus operator (-).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (^). This will call power (.^)!

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mrddivide** ↑

```
obj = mrddivide(obj,another)
```

Description:

Right division operator (/), same as (./).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*). Here assumed equal to .* operator.

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **plus** ↑

obj = plus(obj,another)

Description:

Plus operator (+).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **power** ↑

obj = power(obj,another)

Description:

Power operator (.^).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

```
obj = rdivide(obj,another)
```

Description:

Right division operator (./).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

Square root.

Input:

- obj : An object of class nb_param.

Output:

- obj : An object of class nb_param.

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*).

Input:

- obj : A scalar number, nb_param object or string.
- another : A scalar number, nb_param object, nb_mySD object or string.

Output:

- obj : An object of class nb_param or nb_mySD.

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus.

Input:

- obj : An object of class nb_param.

Output:

- obj : An object of class nb_param.

Written by Kenneth Sæterhagen Paulsen

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus.

Input:

- obj : An object of class nb_param.

Output:

- obj : An object of class nb_param.

Written by Kenneth Sæterhagen Paulsen

■ nb_ss

Go to: [Properties](#) | [Methods](#)

Create a class representing a steady-state variable. Same as a double but indexing like (-d), (d) or (+1) on a scalar object just return the same value as the original data (otherwise indexing works as a normal double).

Superclasses:

double

Constructor:

obj = nb_ss(num)

Input:

- num : A double.

Output:

- obj : A nb_ss object array with same size as num.

Written by Kenneth Sæterhagen Paulsen

Properties:

Methods:

■ nb_st

Go to: [Properties](#) | [Methods](#)

An abstract superclass to make it possible to construct a vector of nb_stTerm, nb_stBase and nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- error
- string
- trend

- **error** ↑

Error message thrown during stationarization.

- **string** ↑

The string representing the new stationarized equation.

- **trend** ↑

The trend growth of this term/base/parameter.

Methods:

- | | | | | |
|----------------------|-----------|-----------|--------------|----------------|
| • bgp | • detrend | • exp | • isTrending | • log |
| • logncdf | • lognpdf | • minus | • mpower | • mrdivide |
| • mtimes | • normcdf | • norminv | • normpdf | • plus |
| • power | • rdivide | • sgp | • sqrt | • steady_state |
| • steady_state_first | • times | • uminus | • uplus | |

► **bgp** ↑

obj = bgp(obj)

Description:

Balanced growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

► **detrend** ↑

obj = detrend(obj)

Description:

De-trend operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

► **exp ↑**

obj = exp(obj)

Description:

Exponential.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.log](#)

Written by Kenneth Sæterhagen Paulsen

► **isTrending ↑**

ret = isTrending(obj)

Description:

Test if nb_st object is trending.

Written by Kenneth Sæterhagen Paulsen

► **log** ↑

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.exp](#)

Written by Kenneth Sæterhagen Paulsen

► **logncdf** ↑

obj = logncdf(obj,m,k)

Description:

Log normal cdf.

Input:

- obj : An object of class nb_st.

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

- k : A parameter such that the mean of the lognormal is $k \exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.lognpdf](#)

Written by Kenneth Sæterhagen Paulsen

► **lognpdf** ↑

obj = lognpdf(obj,m,k)

Description:

Log normal pdf.

Input:

- obj : An object of class nb_st.

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.logncdf](#)

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

obj = minus(obj,another)

Description:

Minus operator (-).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.plus](#), [nb_st.uminus](#)

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (^), which is the same as using .^.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.power](#)

Written by Kenneth Sæterhagen Paulsen

► **mrdivide** ↑

obj = mrdivide(obj,another)

Description:

Right division operator (/), which is the same as using ./.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

obj = mtimes(obj,another)

Description:

Times operator (*), which is the same as using .*.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.times](#)

Written by Kenneth Sæterhagen Paulsen

► **normcdf** ↑

obj = normcdf(obj,m,k)

Description:

Normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

► **norminv** ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

► **normpdf** ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

► **plus** ↑

obj = plus(obj,another)

Description:

Plus operator (+).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.minus](#), [nb_st.uplus](#)

Written by Kenneth Sæterhagen Paulsen

► **power** ↑

obj = power(obj,another)

Description:

Power operator (.^), which is the same as using ^.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.mpower](#)

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

obj = rdivide(obj,another)

Description:

Right division operator (./), which is the same as using /.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.mrdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **sgp** ↑

obj = sgp(obj)

Description:

Stochastic growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.power](#)

Written by Kenneth Sæterhagen Paulsen

► **steady_state** ↑

obj = steady_state(obj)

Description:

Steady-state operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

► **steady_state_first** ↑

obj = steady_state_first(obj)

Description:

Steady-state (first regime) operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

obj = times(obj,another)

Description:

Times operator (.*), which is the same as using *.

Input:

- obj : A scalar number or a nb_st object.

- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.mtimes](#)

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus (-).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.plus](#)

Written by Kenneth Sæterhagen Paulsen

► **uplus** ↑

obj = uplus(obj)

Description:

Unary plus (+).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_st.plus](#)

Written by Kenneth Sæterhagen Paulsen

■ **nb_stParam**

Go to: [Properties](#) | [Methods](#)

A class representing a parameter in a equation during stationarization.

Superclasses:

[nb_st](#), [matlab.mixin.Heterogeneous](#)

See also:

[nb_stTerm](#), [nb_stBase](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [error](#)
- [string](#)
- [trend](#)
- [value](#)

- **error** ↑

Error message thrown during stationarization.

Inherited from superclass NB_ST

- **string** ↑

The string representing the new stationarized equation.

Inherited from superclass NB_ST

- **trend** ↑

The trend growth of this term/base/parameter.

Inherited from superclass NB_ST

- **value** ↑

The parameter value, as a scalar double.

Methods:

- | | | | | |
|--------------------------------------|---------------------------|---------------------------|------------------------------|--------------------------------|
| • bgp | • detrend | • exp | • isTrending | • log |
| • logncdf | • lognpdf | • minus | • mpower | • mrdivide |
| • mtimes | • normcdf | • norminv | • normpdf | • plus |
| • power | • rdivide | • sgp | • sqrt | • steady_state |
| • steady_state_first | • times | • uminus | • uplus | |

► **bgp** ↑

```
obj = bgp(obj)
```

Description:

Balanced growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► [detrend](#) ↑

```
obj = detrend(obj)
```

Description:

De-trend operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► [exp](#) ↑

```
obj = exp(obj)
```

Description:

Exponential.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.log](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► [isTrending](#) ↑

```
ret = isTrending(obj)
```

Description:

Test if nb_st object is trending.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► [log](#) ↑

```
obj = log(obj)
```

Description:

Natural logarithm.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.exp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **logncdf** ↑

obj = logncdf(obj,m,k)

Description:

Log normal cdf.

Input:

- obj : An object of class nb_st.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.lognpdf](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **lognpdf** ↑

obj = lognpdf(obj,m,k)

Description:

Log normal pdf.

Input:

- obj : An object of class nb_st.
- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.
- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.logncdf](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► minus ↑

obj = minus(obj,another)

Description:

Minus operator (-).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.plus](#), [nb_stparam.uminus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► mpower ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (^), which is the same as using \cdot^{\wedge} .

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.power](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **mrdivide** ↑

```
obj = mrdivide(obj,another)
```

Description:

Right division operator (/), which is the same as using $\cdot /$.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.rdivide](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **mtimes** ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*), which is the same as using .*.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.times](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **normcdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **norminv** ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **normpdf** ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **plus** ↑

```
obj = plus(obj,another)
```

Description:

Plus operator (+).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.minus](#), [nb_stparam.uplus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **power** ↑

```
obj = power(obj,another)
```

Description:

Power operator (.^), which is the same as using ^.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.mpower](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **rdivide** ↑

obj = rdivide(obj,another)

Description:

Right division operator (./), which is the same as using /.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.mrdivide](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **sgp** ↑

obj = sgp(obj)

Description:

Stochastic growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.power](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **steady_state** ↑

obj = steady_state(obj)

Description:

Steady-state operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **steady_state_first** ↑

obj = steady_state_first(obj)

Description:

Steady-state (first regime) operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **times** ↑

obj = times(obj,another)

Description:

Times operator (.*), which is the same as using *.

Input:

- obj : A scalar number or a nb_st object.

- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.mtimes](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus (-).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.plus](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **uplus** ↑

obj = uplus(obj)

Description:

Unary plus (+).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stparam.plus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

■ nb_stTerm

Go to: [Properties](#) | [Methods](#)

A class representing a trending or non-trending variable/term during stationarization.

Superclasses:

nb_st, matlab.mixin.Heterogeneous

Constructor:

```
obj = nb_stTerm(varName,trend,leadOrLag)
```

Input:

- varName : A one line char with the name of the base variable, or a cellstr with a set of base variables.
- trend : A double matching the number of base variables given. It must store the trend growth of each base variable. Set to 0 to indicate a non-trending variable.
- leadOrLag : A double matching the number of base variables given. It must store the number of periods the base value is leaded or lagged.
- unitRoot : A logical matching the number of base variables given. Give true for the elements that are unit root variables. Default is false. If a scalar logical is given it will be applied to all base variables.

Output:

- obj : A vector of nb_stTerm objects.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- error • string • trend

- **error** ↑

Error message thrown during stationarization.

Inherited from superclass NB_ST

- **string** ↑

The string representing the new stationarized equation.

Inherited from superclass NB_ST

- **trend** ↑

The trend growth of this term/base/parameter.

Inherited from superclass NB_ST

Methods:

- bgp
- logncdf
- mtimes
- power
- steady_state_first
- detrend
- lognpdf
- normcdf
- rdivide
- times
- exp
- minus
- norminv
- sgp
- uminus
- isTrending
- mpower
- normpdf
- sqrt
- uminus
- log
- mrdivide
- plus
- steady_state
- uplus

► **bgp** ↑

obj = bgp(obj)

Description:

Balanced growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **detrend** ↑

obj = detrend(obj)

Description:

De-trend operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **exp ↑**

obj = exp(obj)

Description:

Exponential.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.log](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **isTrending ↑**

ret = isTrending(obj)

Description:

Test if nb_st object is trending.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **log** ↑

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.exp](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **logncdf** ↑

obj = logncdf(obj, m, k)

Description:

Log normal cdf.

Input:

- obj : An object of class nb_st.

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.lognpdf](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **lognpdf** ↑

obj = lognpdf(obj,m,k)

Description:

Log normal pdf.

Input:

- obj : An object of class nb_st.

- m : A parameter such that the mean of the lognormal is $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

- k : A parameter such that the mean of the lognormal is k $\exp((m+k^2)/2)$. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.logncdf](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **minus** ↑

obj = minus(obj,another)

Description:

Minus operator (-).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.plus](#), [nb_stterm.uminus](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (^), which is the same as using .^.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.power](#)

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **mrddivide** ↑

obj = mrddivide(obj,another)

Description:

Right division operator (/), which is the same as using ./.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.rdivide](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **mtimes** ↑

obj = mtimes(obj,another)

Description:

Times operator (*), which is the same as using .*.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.times](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **normcdf** ↑

```
obj = normcdf(obj,m,k)
```

Description:

Normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **norminv** ↑

```
obj = norminv(obj,m,k)
```

Description:

Inverse normal cdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth SÃ¶terhagen Paulsen

Inherited from superclass NB_ST

► **normpdf** ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : An object of class nb_st.
- m : The mean of the distribution. Either as a scalar double or a nb_stParam object.
- k : The std of the distribution. Either as a scalar double or a nb_stParam object.

Output:

- obj : An object of class nb_stTerm or nb_stParam.

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **plus ↑**

```
obj = plus(obj,another)
```

Description:

Plus operator (+).

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.minus](#), [nb_stterm.uplus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **power ↑**

```
obj = power(obj,another)
```

Description:

Power operator ($.^$), which is the same as using $^$.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.mpower](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **rdivide** ↑

```
obj = rdivide(obj,another)
```

Description:

Right division operator ($./$), which is the same as using $/$.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.mrdivide](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **sgp** ↑

obj = sgp(obj)

Description:

Stochastic growth path operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.power](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **steady_state** ↑

```
obj = steady_state(obj)
```

Description:

Steady-state operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **steady_state_first** ↑

```
obj = steady_state_first(obj)
```

Description:

Steady-state (first regime) operator.

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **times** ↑

```
obj = times(obj,another)
```

Description:

Times operator (.*), which is the same as using *.

Input:

- obj : A scalar number or a nb_st object.
- another : A scalar number or a nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.mtimes](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **uminus** ↑

```
obj = uminus(obj)
```

Description:

Unary minus (-).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.plus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus (+).

Input:

- obj : A nb_st object.

Output:

- obj : An object of class nb_stParam or nb_stTerm.

See also:

[nb_stTerm](#), [nb_stParam](#), [nb_stterm.plus](#)

Written by Kenneth Sæterhagen Paulsen

Inherited from superclass NB_ST

■ **nb_symMatrix**

Go to: [Properties](#) | [Methods](#)

An object that represent a matrix of symbolic parameters of a matrix.

The values of the symbols may be provided. Some methods may need these values to be used.

Constructor:

```
obj = nb_symMatrix(input1, input2)
```

Input:

> Alternativ 1:

- input1 : The number of rows of the symbolic matrix.
- input2 : The number of cols of the symbolic matrix.

> Alternativ 2:

- input1 : A rows x cols cellstr with the symbols, or a rows x 1 char (which will be converted to a rows x 1 cellstr).

> Alternative 3:

- input1 : A rows x cols numeric matrix.

Output:

> Alternativ 1:

- obj : A rows x cols nb_symMatrix object with all elements set to zero.

> Alternativ 2 and 3:

- obj : A rows x cols nb_symMatrix object.

Written by Kenneth Sæterhagen Paulsen

Properties:

• symbols

- **symbols** ↑

A rows x cols nb_term object with the symbolic representation of the matrix.

Methods:

• cellstr	• disp	• eq	• exp	• horzcat	• log
• minus	• mpower	• mrdivide	• mtimes	• plus	• power
• rdivide	• rref	• sqrt	• subsasgn	• subsref	• times
• uminus	• uplus	• vertcat			

► **cellstr** ↑

c = cellstr(obj)

Description:

Display object (In the command window)

Input:

- obj : An object of class nb_symMatrix.

Output:

The object displayed on the command line.

Written by Kenneth S. Paulsen

► **disp** ↑

```
disp(obj)
```

Description:

Display object (In the command window)

Input:

- obj : An object of class nb_symMatrix.

Output:

The object displayed on the command line.

Written by Kenneth S. Paulsen

► **eq ↑**

```
ret = eq(obj,another)
```

Description:

Equality operator (==) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

Written by Kenneth Sætherhagen Paulsen

► **exp ↑**

```
obj = exp(obj)
```

Description:

Exponential.

Input:

- obj : A nb_symMatrix object.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.log](#)

Written by Kenneth Sæterhagen Paulsen

► **horzcat** ↑

obj = horzcat(a,b,varargin)

Description:

Horizontal concatenation of nb_symMatrix objects ([a,b]).

Input:

- a : An object of class nb_symMatrix.
- varargin : Optional number of nb_symMatrix objects.

Output:

- obj : An nb_symMatrix object.

Examples:

```
obj = [a,b];
obj = [a,b,c];
```

See also:

[nb_symMatrix.vertcat](#)

Written by Kenneth S. Paulsen

► **log** ↑

obj = log(obj)

Description:

Natural logarithm.

Input:

- obj : A nb_symMatrix object.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.exp](#)

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

obj = minus(obj,another)

Description:

Minus operator (-) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.plus](#), [nb_symMatrix.uminus](#)

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

obj = mpower(obj,another)

Description:

Power operator (^), which is the same as using .^.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.power](#)

Written by Kenneth Sæterhagen Paulsen

► **mrddivide** ↑

obj = mrddivide(obj,another)

Description:

Right division operator (/), which is the same as using ./.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **mtimes** ↑

obj = mtimes(obj,another)

Description:

Times operator (*), which is the same as using .*.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.times](#)

Written by Kenneth Sæterhagen Paulsen

► **plus** ↑

obj = plus(obj,another)

Description:

Plus operator (+) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.minus](#), [nb_symMatrix.uplus](#)

Written by Kenneth Sæterhagen Paulsen

► **power** ↑

obj = power(obj,another)

Description:

Division operator ($.^$) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.power](#)

Written by Kenneth Sæterhagen Paulsen

► **rdivide** ↑

obj = rdivide(obj,another)

Description:

Division operator ($./$) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.times](#)

Written by Kenneth Sæterhagen Paulsen

► **rref** ↑

obj = rref(obj)

Description:

This method produces the row echelon form of A.

Input:

- obj : A NxM nb_symMatrix object.

Output:

- obj : A NxM nb_symMatrix object.

Examples:

See also:

[nb_rref](#)

Written by Kenneth Sæterhagen Paulsen

► **sqrt** ↑

obj = sqrt(obj)

Description:

Square root.

Input:

- obj : A nb_symMatrix object.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.power](#)

Written by Kenneth Sæterhagen Paulsen

► **subsasgn** ↑

varargout = subsasgn(obj,s,b)

Description:

Makes it possible to do subscripted assignment an object of class nb_symMatrix.

Input:

- obj : An object of class nb_symMatrix.
- s : The same input as when you do subscripted assignment one a double matrix
- b : The assign data at the given index s. Either numeric or an object of class nb_symMatrix.

Output:

- obj : An object of class nb_symMatrix.

Examples:

```
obj      = nb_symMatrix(2,2);
obj(1,1) = nb_symMatrix('X',1);
```

Written by Kenneth S. Paulsen

► **subsref ↑**

```
varargout = subsref(obj,s)
```

Description:

Makes it possible to do subscripted reference to an object of class nb_symMatrix.

Input:

- obj : An object of class nb_symMatrix.
- s : A structure on how to index the object.

Output:

- obj : An object of class nb_symMatrix.

Written by Kenneth S. Paulsen

► **times ↑**

```
obj = times(obj,another)
```

Description:

Times operator (.*) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object or double.
- another : A nb_symMatrix object or double.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.rdivide](#)

Written by Kenneth Sæterhagen Paulsen

► **uminus** ↑

obj = uminus(obj)

Description:

Unary minus operator (-) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.minus](#)

Written by Kenneth Sæterhagen Paulsen

► **uplus** ↑

obj = uplus(obj)

Description:

Unary plus operator (+) for nb_symMatrix objects.

Input:

- obj : A nb_symMatrix object.

Output:

- obj : A nb_symMatrix object.

See also:

[nb_symMatrix.plus](#)

Written by Kenneth Sæterhagen Paulsen

► **vertcat ↑**

obj = vertcat(a,b,varargin)

Description:

Vertical concatenation of nb_symMatrix objects ([a;b]).

Input:

- a : An object of class nb_symMatrix.
- varargin : Optional number of nb_symMatrix objects.

Output:

- obj : An nb_symMatrix object.

Examples:

```
obj = [a;b];
obj = [a;b;c];
```

See also:

[nb_symMatrix.horzcat](#)

Written by Kenneth S. Paulsen

■ **nb_term**

Go to: [Properties](#) | [Methods](#)

The superclass of all mathematical terms included in an equation.

Be aware that you still cannot call any method on a vector of any of them!

Superclasses:

matlab.mixin.Heterogeneous

Subclasses:

nb_base, nb_num, nb_equation

See also:

[nb_base](#), [nb_num](#), [nb_equation](#)

Written by Kenneth Sæterhagen Paulsen

Properties:

- [numberOfTerms](#)
- [operator](#)
- [terms](#)

- [numberOfTerms](#) ↑

The number of terms. As a scalar integer.

- [operator](#) ↑

Type of operator splitting the terms.

- [terms](#) ↑

A vector of nb_term objects.

Methods:

- | | | | | |
|-------------------------------|--------------------------------|-----------------------------|----------------------------|----------------------------|
| • cellstr | • clean | • correct | • exp | • gap |
| • generalFunc | • initialize | • intersect | • ismember | • log |
| • logncdf | • lognpdf | • minus | • mpower | • mrdivide |
| • mtimes | • normcdf | • norminv | • normpdf | • plus |
| • power | • rdivide | • removePar | • simplify | • split |
| • sqrt | • steady_state | • times | • toString | • uminus |
| • union | • unique | • uplus | | |

► [cellstr](#) ↑

c = [toString](#)(obj)

Description:

Convert array of nb_term objects to cellstr.

Inputs:

- obj : An array of nb_term objects.

Output:

- c : A cellstr with same size as obj.

See also:

[toString](#)

Written by Kenneth Sæterhagen Paulsen

► **clean** ↑

obj = clean(obj)

Description:

Clean up nb_term object after operations.

Written by Kenneth Sæterhagen Paulsen

► **correct** ↑

expr = nb_term.correct(expr)

Description:

Correct term for .*, ./ and .^ operators.

Input:

- expr : A one line char with a mathematical expression.

Output:

- expr : Corrected version of the input.

Written by Kenneth Sæterhagen Paulsen

► **exp** ↑

```
obj = exp(obj)
```

Description:

exp operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **gap** ↑

```
obj = gap(obj)
```

Description:

Gap operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **generalFunc** ↑

```
obj = generalFunc(obj, func, varargin)
```

Description:

Call general function on a nb_term objects.

Input:

- obj : A vector of nb_term objects.
- func : A one line char with the function to apply on the nb_term objects.

Optional input:

- nb_term objects representing the extra inputs to the function.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

► initialize ↑

```
obj = nb_term.initialize(rows,cols)
```

Description:

Initialize vector of nb_term objects.

Input:

- rows : Number of rows. As a scalar integer.
- cols : Number of columns. As a scalar integer.

Output:

- obj : A rows x cols nb_term object.

Written by Kenneth SÃ¶terhagen Paulsen

► intersect ↑

```
int = intersect(obj,another)
```

Description:

Get the intersect of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- int : The intersect, as a vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

► **ismember** ↑

```
[ind,loc] = ismember(obj,another)
```

Description:

Check if the terms in obj is member of the terms in another.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- ind : A logical vector with same size as obj. Elements that are true are found to be in the another input.
- loc : A double vector with same size as obj. Locations of the elements that are found in the another input. If not found 0 is returned.

Written by Kenneth SÃ¶terhagen Paulsen

► **log** ↑

```
obj = log(obj)
```

Description:

log operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **logncdf** ↑

obj = logncdf(obj,m,k)

Description:

Log-normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **lognpdf** ↑

obj = lognpdf(obj,m,k)

Description:

Log-normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **minus** ↑

```
obj = minus(obj,another)
```

Description:

Minus operator (-) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

► **mpower** ↑

```
obj = mpower(obj,another)
```

Description:

Power operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sætherhagen Paulsen

► mrdivide ↑

```
obj = mrdivide(obj,another)
```

Description:

Division operator (/) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sætherhagen Paulsen

► mtimes ↑

```
obj = mtimes(obj,another)
```

Description:

Times operator (*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

► **normcdf** ↑

obj = normcdf(obj,m,k)

Description:

Normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **norminv** ↑

obj = norminv(obj,m,k)

Description:

Inverse normal cdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

► normpdf ↑

```
obj = normpdf(obj,m,k)
```

Description:

Normal pdf.

Input:

- obj : A vector of nb_term objects.
- m : The mean of the distribution, as an object of class nb_term.
- k : The std of the distribution, as an object of class nb_term.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

► plus ↑

```
obj = plus(obj,another)
```

Description:

Plus operator (+) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sætherhagen Paulsen

► power ↑

```
obj = power(obj,another)
```

Description:

Power operator ($.^$) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sætherhagen Paulsen

► rdivide ↑

```
obj = rdivide(obj,another)
```

Description:

Division operator ($./$) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.
- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

► **removePar** ↑

```
expr = removePar(expr)
```

Description:

Is the expression enclosed with parentheses?

Written by Kenneth Sæterhagen Paulsen

► **simplify** ↑

```
expr = nb_term.simplify(expr)
expr = nb_term.simplify(expr,waitbar)
```

Description:

Simplify the mathematical expressions.

Input:

- expr : A one line char with a mathematical expression or a cellstr with the mathematical expressions.
- waitbar : A nb_waitbar5 object. It will use the next available waitbar of the given object. If empty no waitbar is added.

Output:

- obj : A cellstr with the simplified mathematical expressions.

Written by Kenneth Sæterhagen Paulsen

► **split** ↑

```
obj = nb_term.split(expr)
obj = nb_term.split(expr,waitbar)
```

Description:

Split the mathematical expression into separate terms.

Input:

- expr : A one line char with a mathematical expression.
- waitbar : A nb_waitbar5 object. If empty no waitbar is added.

Output:

- obj : A nb_term object representing the mathematical expression.

Written by Kenneth SÃ¶terhagen Paulsen

► **sqrt** ↑

```
obj = sqrt(obj)
```

Description:

sqrt operator for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth SÃ¶terhagen Paulsen

► **steady_state** ↑

```
obj = steady_state(obj)
```

Description:

Steady-state operator.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **times** ↑

obj = times(obj,another)

Description:

Times operator (.*) for nb_term objects.

Input:

Two cases:

One input:

- obj : A vector of nb_term objects.

Two inputs:

- obj : A scalar nb_term object.

- another : A vector of nb_term objects.

Output:

- obj : A scalar nb_term object.

Written by Kenneth Sæterhagen Paulsen

► **toString** ↑

string = toString(anyObject)

Description:

Converts object into a string.

Input:

- anyObject : A double, logical, nb_ts, nb_cs, nb_data or cell.

Output:

- string : A string

See also:

[num2str](#), [int2str](#), [nb_cellstr2String](#), [nb_any2String](#)

Written by Kenneth Sæterhagen Paulsen

► uminus ↑

obj = uminus(obj)

Description:

Unary minus operator (-) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term object.

Written by Kenneth Sæterhagen Paulsen

► union ↑

uni = union(obj,another)

Description:

Get the union of two vectors of nb_term objects.

Input:

- obj : A vector of nb_term objects.
- another : A vector of nb_term objects.

Output:

- uni : The union, as a vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

► **unique** ↑

```
uniq = unique(obj,another)
```

Description:

Locate the unique nb_term objects in a vector of nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- uniq : The unique nb_term objects.

- ind : Vector such that uniq = obj(ind).

Written by Kenneth Sæterhagen Paulsen

► **uplus** ↑

```
obj = uplus(obj)
```

Description:

Unary plus opertor (+) for nb_term objects.

Input:

- obj : A vector of nb_term objects.

Output:

- obj : A vector of nb_term objects.

Written by Kenneth Sæterhagen Paulsen

■ **nb_writeHelp**

Go to: [Properties](#) | [Methods](#)

A class for writing the help for the estimator packages and model classes of the NB Toolbox.

Constructor:

```
obj = nb_writeHelp(name,option,typeName,dataType)

Input:

- name      : The name of the class or package to document.

- option    : Either the name of the option or 'all'.

- typeName : Indicate if it is a class ('class') or package
              ('package').

- dataType : Either 'timeSeries', 'crossSectional' or 'both'.
```

Output:

```
- obj      : An object of class nb_writeHelp.
```

Examples:

Written by Kenneth Sætherhagen Paulsen

Properties:

- [dataType](#)
- [helpStruct](#)
- [max](#)
- [nameOfPOrC](#)
- [option](#)

- [options](#)
- [typeName](#)

- [dataType](#) ↑

Either 'timeSeries', 'crossSectional' or 'both'.

- [helpStruct](#) ↑

A struct that stores the help text of all the options.

- [max](#) ↑

Max number of chars in the printed doc.

- [nameOfPOrC](#) ↑

Name of the package or class.

- [option](#) ↑

The option to get the help on, or 'all'.

- [options](#) ↑

A cellstr with the names of the options that the

- [typeName](#) ↑

Indicate if it is a class or package

Methods:

- help
 - set
-

► **help** ↑

```
helpText = help(obj)
```

Description:

Get the help given the property values of the object.

Input:

- obj : An object of class nb_writeHelp.

Output:

- helpText : A char with the help on the package or class.

Written by Kenneth SÅ!terhagen Paulsen

► **set** ↑

```
obj = set(obj,varargin)
```

Description:

Sets the properties of an object of class nb_writeHelp. Give the property name as a string. The input that follows should be the value you want to assign to the property.

Multiple properties can be set with one method call.

Input:

- obj : An object of class nb_writeHelp.
- varargin : Every odd input must be a string with the name of the property wanted to be set. Every even input must be the value you want to set the given property to.

Output:

- obj : The same object of class nb_writeHelp with the given properties reset

Examples:

```
obj.set('propertyName',PropertyValue,...);
```

Written by Kenneth Sæterhagen Paulsen

◆ nb_bartlettKernel

```
y = nb_bartlettKernel(x)
```

Description:

Barlett kernel

Input:

- x : A double

Output:

- y : A double

Written by Kenneth Sæterhagen Paulsen

◆ nb_calculateMoments

```
[m,c] = nb_calculateMoments(A,B,C,vcv,mX,varX,nLags,type,tol,maxIter)
```

Description:

Calculate moments given the companion form of the model:

$y_t = Ay_{t-1} + Bx_t + Ce_t$, where $e_t \sim N(0, vcv)$ and
 $x_t \sim PHI(mX, varX)$

Input:

- A,B,C,vcv,,mX,varX : See description of function.
- nLags : The number of autocorrelation/autovariances to calculate.
- type : Either 'covariance' or 'correlation'.
- tol : The tolerance of the iteration procedure of the lyapunov equation. Default is $\text{eps}^{(1/3)}$.
- maxiter : Maximum number of iterations of the iteration procedure of the lyapunov equation.. Default is 1000.

Output:

- m : Unconditional mean of the model. As a nVar x 1 double.
- c : Unconditional (auto)covarianse/correlation matrix. As a nVar x nVar x nLags + 1 double.

See also:

[nb_model_generic.theoreticalMoments](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_cell2func

func = nb_cell2func(eqs,inputs)

Description:

Convert equations to function handle.

Input:

- eqs : A cellstr with the equations.
- inputs : A one line char with the inputs to the created function handle. E.g. '(x,y)'.

Output:

- func : A function handle.

Written by Kenneth Sæterhagen Paulsen

◆ nb_coeff2excel

◆ nb_constructInputPar

inputs = nb_constructInputPar(vars,pars)

Description:

Construct inputs enclosed in parenthesis. E.g. '(var1,var2,par1,par2)'.

Input:

- vars : A cellstr with the variables.
- pars : A cellstr with the parameters.

Output:

- inputs : A one line char. See description.

See also:

[nb_cell2func](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_constructStackedCorrelationMatrix**

`sigmaF = nb_constructStackedCorrelationMatrix(c)`

Description:

Construct the stacked correlation matrix.

Input:

- c : A nVar x nVar x nLags + 1 x nPage double storing the (auto)correlation matrices. The first page stores the contemporaneous correlation matrix, second the autocorrelation matrix with lag 1 and so on.

Output:

- sigmaF : The stacked correlation matrix. As a nVar*(nLags+1) x nVar*(nLags+1) x nPage double.

See also:

[nb_calculateMoments](#), [nb_model_generic.theoreticalMoments](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_cov2corr**

`c = nb_cov2corr(c)`

Description:

Go from covariance to correlation matrix.

Input:

- c : Page one is the contemporaneous correlation matrix. Page 2:end is the autocorrelation of degree ≥ 1 .

Output:

- c : Correlation matrix.

Written by Kenneth Sæterhagen Paulsen

◆ nb_createGenericNames

```
[out,outS,inS,reorder] = nb_createGenericNames(in,generic)
```

Description:

Create generic names index by integers, e.g. {'x(1)';'x(2)'}

Input:

- in : A cellstr with the original names.
- generic : The generic name, e.g. 'x'.

Output:

- out : A cellstr. See description.
- outS : in is sorted and out is index by the reordering.
- inS : The sorted version of in.
- reorder : Reordering index.

Written by Kenneth Sæterhagen Paulsen

◆ nb_doSymbolic

```
[derivFunc,I,J,V,jac] = nb_doSymbolic(fh,vars,varValues,pars,pValues)
```

Description:

Construct function handle with the derivatives of a given function, or evaluate the full jacobian.

Input:

- fh : A function handle. @(x,y)x*y or @(x,y,p)x*y^p. Please order parameters at the end!
- vars : List the variables input to the function. E.g. {'x','y'}.
- pars : List the parameters input to the function. E.g. {} or {'p'}.
- varValues : The values of the variables for where you want to evaluate the derivative. Only needed if the output V and/or jac are

asked for. Must have same length as vars input.

- pValues : The values of the parameters. Only needed if the output V and/or jac are asked for. Must have same length as pars input.

Output:

- derivFunc : Function handle with the symbolic derivatives.
- I,J,V : Vectors such that jac = sparse(I,J,V,nEqs,nVar), where nEqs is the number of equations and nVar is the number of variables.
- jac : The evaluated jacobian at the provided values. varValues and pValues must be given (also applies to V)!

Examples:

```
func          = @(x,y)x*y;
[fH,I,J,V,jac] = nb_doSymbolic(func,{'x','y'},[],[2,1]);
func          = @(x,y,p)x*y^p; % Parameters at the end!
[fH2,I2,J2,V2,jac2] = nb_doSymbolic(func,{'x','y'},{'p'},[2,1],0.5);
```

See also:

[nb_mySD](#)

Written by Kenneth Sæterhagen Paulsen

◆ **[nb_excel2coeff](#)**

◆ **[nb_funcWrapper](#)**

```
value = nb_funcWrapper(x,fHandle,inputs)
```

Description:

Create a wrapper function to assign optional inputs to a function handle.

Useful for speeding up function evaluation during parfor with use of feval.

Caution: For speed, the input are not tested!

Input:

- x : Main input to function.
- fHandle : A function handle.
- inputs : A cell array with the inputs passed to the function handle.

Output:

- value : The output of the function.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_functionHandle2Struct**

```
s = nb_functionHandle2Struct(f)
```

Description:

Convert function handle to struct.

Input:

- f : function_handle to convert to struct.

Output:

- s : A struct representing the function_handle

Examples:

```
s = 2;
s2 = 3;
f = @(x)x + s + s2
s = nb_functionHandle2Struct(f)
```

See also:

[nb_struct2functionHandle](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_getModelListFromLibrary**

```
modelList = nb_getModelListFromLibrary(folder)
```

Description:

Get model list from library folder.

Input:

- folder : A full absolute path to the model library.

Output:

- modelList : A list of all models stores as .mat files.

Written by Erlend Salvesen Njålstad

◆ nb_getSymbolicDerivIndex

```
[I,J] = nb_getSymbolicDerivIndex(symDeriv,all,derivEqs)
```

Description:

Find sparse indexes when utilizing the nb_mySD class and want to construct the full jacobian.

Input:

- symDeriv : A vector of nb_mySD objects.
- all : Either a cellstr with all bases of all equations or a one line char with the generic name used when using nb_mySD.
- derivEqs : The derivatives as a cellstr. If empty [symDeriv.derivatives] will be used.

Output:

- I,J : Sparse indexes that can be used to initialize the sparse jacobian.

Written by Kenneth Sæterhagen Paulsen

◆ nb_gradient

```
G = nb_gradient(func,x,h)
```

Description:

Calculate the gradient of the multivariate function func at the point x using a step length h.

See Abramowitz and Stegun (1965) formulas 25.3.21.

Input:

- func : A MATLAB function handle
- x : The point of evaluation. As a n x 1 double.
- h : Step length. Either a 1x1 double or a n x 1 double. Default is abs(x)*eps^(1/6).

Output:

G : A n x 1 double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_hessian

H = nb_hessian(func,x,h)

Description:

Calculate the hessian of the multivariate function func at the point x using a step length h.

See Abramowitz and Stegun (1965) formulas 25.3.23 and 25.3.27.

Input:

- func : A MATLAB function handle
- x : The point of evaluation. As a n x 1 double.
- h : Step length. Either a 1x1 double or a n x 1 double. Default is abs(x)*eps^(1/6).

Output:

H : A n x n double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_interpretPerc

newPerc = nb_interpretPerc(perc,inverse)

Description:

Input:

- perc : Depend on the inverse input.
 - inverse :
- > inverse == true : Input must be a 1 x N double with percentiles in the format [5,15,25,35,65,75,85,95], while the output is percentiles in the format [0.3,0.5,0.7,0.9].
- > inverse == false : Switch input and output from the inverse == true case.

Output:

- newPerc : Depend on the inverse input.

Written by Kenneth Sætherhagen Paulsen

◆ nb_isStructOfFunctionHandle

```
ret = nb_isStructOfFunctionHandle(structThatRepresentsTheFunctionHandle)
```

Description:

Is the struct representing a function_handle object?

Input:

- structThatRepresentsTheFunctionHandle : A struct which is an output from the nb_functionHandle2Struct function.

Output:

- ret : true or false.

See also:

[nb_functionHandle2Struct](#), [nb_struct2functionHandle](#)

Written by Kenneth Sætherhagen Paulsen

◆ nb_jacobian

```
JAC = nb_jacobian(func,x,y)
```

Description:

Calculate the jacobian of the multivariate function func at the point x using a step length x - w. The following formula is used:

```
JAC(:,j) = (func(x(1),...,x(j),y(j+1),...,y(m)) -  
func(x(1),...,x(j-1),y(j),...,y(m)))/(x(j) - y(j))
```

Input:

- func : A MATLAB function handle
- x : The point of evaluation. As a m x 1 double.
- y : x + step length. As a m x 1 double.

Output:

G : A m x m double.

Written by Kenneth Sæterhagen Paulsen

◆ nb_latinHypercubeSim

V = nb_latinHypercubeSim(nPoints, nVars)

Description:

Simulating optimized draws from the Latin hypercube from the uniform distribution to be used in monte-carlo integration or other such applications.

Algorithm is taken from:

Larson et. al. (2005) "Supplying Local Microphysics Parameterizations with Information about Subgrid Variability: Latin Hypercube Sampling"

See section 3b.

Input:

- nPoints : Number of points of the desired Latin hypercube (LH).
- nVars : Number of variables in the LH.

Output:

- V : A nPoints x nVars double.

See also:

[nb_monteCarloSim](#)

Written by Kenneth Sæterhagen Paulsen

◆ nb_loadModelsFromLibrary

models = nb_loadModelsFromLibrary(folder)

Description:

Load models from library folder.

Input:

- folder : A full absolute path to the model library.

Output:

- models : A vector of nb_model_update_vintages.

Written by Kenneth Sæterhagen Paulsen

◆ nb_maxDeriv

```
value = nb_maxDeriv(expr1,expr2,deriv1,deriv2)
```

Description:

Calculate the derivative of the max(expr1,expr2) function. The derivative is equal to the deriv1 if expr1 > expr2, equal to the deriv2 if expr1 < expr2, 1 when expr1 lim(+) -> expr2 and 0 when expr1 lim(-) -> expr2.

Input:

- expr1 : Value of expression 1. A vector with length N or scalar.
- expr2 : Value of expression 2. A vector with length N or scalar.
- deriv1 : Value of the derivative of expression 1. A vector with length N or scalar.
- deriv2 : Value of the derivative of expression 1. A vector with length N or scalar.

Output:

- value : Value of the derivative after the max function is applied.

Written by Kenneth Sæterhagen Paulsen

◆ nb_minDeriv

```
value = nb_minDeriv(expr1,expr2,deriv1,deriv2)
```

Description:

Calculate the derivative of the min(expr1,expr2) function. The derivative is equal to the deriv1 if expr1 < expr2, equal to the deriv2 if expr1 > expr2, 1 when expr2 lim(+) -> expr1 and 0 when expr2 lim(-) -> expr1.

Input:

- expr1 : Value of expression 1. A vector with length N or scalar.
- expr2 : Value of expression 2. A vector with length N or scalar.
- deriv1 : Value of the derivative of expression 1. A vector with length N or scalar.
- deriv2 : Value of the derivative of expression 1. A vector with length N or scalar.

Output:

- value : Value of the derivative after the min function is applied.

Written by Kenneth Sæterhagen Paulsen

◆ **nb_monteCarloSim**

X = nb_monteCarloSim(draws, numVar, method)

Description:

Make draws for monte carlo integration or other usch application.

Input:

- draws : The number of draws from the N-th uniform cube.

- lb : A 1 x N double with the lower bounds.

- ub : A 1 x N double with the upper bounds.

- method : > 'latin' : Uses nb_latinHypercubeSim.
 > 'sobol' : Uses sobolset (part of the MATLAB statistics
 package).
 > 'halton' : Uses haltonset (part of the MATLAB statistics
 package).

Output:

- X : As a draws x N double.

See also:

[nb_latinHypercubeSim](#)

Written by Kenneth Sæterhagen Paulsen

◆ **nb_parzenKernel**

y = nb_parzenKernel(x)

Description:

Barlett kernel

Input:

- x : A double

Output:

- y : A double

Written by Kenneth Sæterhagen Paulsen

◆ nb_quadraticSpectralKernel

```
y = nb_quadraticSpectralKernel(x)
```

Description:

Quadratic spectral kernel

Input:

- x : A double

Output:

- y : A double

Written by Kenneth Sæterhagen Paulsen

◆ nb_shortestpath

```
[dist,path,pred] = nb_shortestpath(A,u,e)
```

Description:

Calculate shortest distance and path between two vertices (nodes) in the graph A.

Input:

- A : A double or sparse double of size N x N.
- u : The vertix (node) to start from. As an scalar integer.
- e : The vertix (node) to end at. As an scalar integer.

Output:

- dist : A scalar double with the shortest distance between the vertices (nodes). If vertex e is not reachable from u inf will be returned.
- path : The shortes path between the two vertices. As a 1 x nSteps double. If vertex e is not reachable from u [] is returned.
- pred : The predecessors along the shortest path.

Examples:

```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W);

[dist,path,pred] = nb_shortestpath(DG,1,6)
```

See also:

[gaimc.bfs](#), [gaimc.dijkstra](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_struct2functionHandle**

```
f = nb_struct2functionHandle(structThatRepresentsTheFunctionHandle)
```

Description:

Convert struct to function_handle.

Input:

- structThatRepresentsTheFunctionHandle : A struct which is an output from the nb_functionHandle2Struct function.

Output:

- f : A function_handle.

See also:

[nb_functionHandle2Struct](#), [nb_isStructOfFunctionHandle](#)

Written by Kenneth Sølnerhagen Paulsen

◆ **nb_testForecastRestrictions**

```
probability = nb_testForecastRestrictions(data,startDate,variables,...  
                                         restrictions)
```

Description:

Calculate probability of passing all forecast restrictions

Input:

```

- data          : Simulation data as a 3D double
                  (nHorizons x nVariables x nDraws)

- startDate    : First date of the data as a string or nb_date object

- variables    : The variables of the data as a cell array of strings

- restrictions : Restrictions as a n x 3 cell array,
                  {variable, date, test}.

    - variable : The variable(s) to test,
                  as a string or cell array of strings.

    - date     : The date as a string. If a cell array is given, a copy
                  of the restriction will be made for every given date.

    - test      : Restriction test as a function handle. The value(s) of
                  the variable(s) in 'variable' at time 'date' are passed
                  as arguments. Should return a logical.

```

Output:

```
- out : The estimated probability as a double
```

Example:

```

restrictions = {'QSA_URR', {'2014Q1', '2014Q2'}, @(x) x < 0; ...
                {'QSA_URR', 'QSA_DPQ_CP'}, '2014Q1', @(x, y) x < y};
probability = nb_testForecastRestrictions(...,
    data, '2010Q1', {'QSA_URR', 'QSA_DPQ_CP'}, restrictions);

```

See also
nb_model_generic.testForecastRestrictions

Written by Henrik Halvorsen Hortemo

◆ **nb_writePostMacroFile**

```
nb_writePostMacroFile(parsedFile, filename)
```

Description:

Write post macro processor model file to a file with extension nbm file.

Input:

```

- parsedFile : The parsed model as a N x 3 cell.

- filename   : File to write the model file to. Extension will be
                  automatically set to .nbm.

```

Written by Kenneth Sæterhagen Paulsen