IRIS

MACROECONOMIC MODELING TOOLBOX

Reference Manual

Release 20151013

by

IRIS Solutions Team

October 13, 2015

Preface

IRIS is a Matlab® toolbox for macroeconomic modeling, developed by the IRIS Solutions Team since 2001. The toolbox is free and open source, distributed under BSD license terms. The current version of IRIS runs in Matlab R2014a or later; we though always recommend to update Matlab to the most recent version.

IRIS has been designed as an integrated software package to support the development and operation of macroeconomic models and model-based analytical systems. To that end, the toolbox integrates core modeling functions (such as a flexible model file language, tools for simulation, estimation, forecasting, or model diagnostics) with supporting infrastructure (such as time series analysis, data management, or reporting) in a user-friendly command-oriented environment.

This document is a reference manual, not a user's guide or tutorial. Check out the website, `www.iris-toolbox.com`, for a number of tutorials covering a range of different topics.

The IRIS Solutions Team currently includes

- *Jaromír Beneš* (International Monetary Fund), team member since 2001.
- *Michael Johnston* (New Zealand Treasury), team member since 2013.
- *Sergey Plotnikov* (OGResearch), team member since 2013.

Contents

Part I —
IRIS Sessions

1 Installing IRIS

Requirements

- Matlab R2014a or later.

Optional components

- The Optimization Toolbox is needed to compute the steady state of non-linear models, and to run estimation.
- LaTeX is a free typesetting system used to produce PDF reports in IRIS.

Installing IRIS

1. Download the latest IRIS zip archive, `IRIS_Tbx_YYYYMMDD.zip`, from the download area on `www.iris-toolbox.com`, and save it in a temporary location on your disk.
2. If you are going to install IRIS in a folder where an older version already resides, completely delete the old version first.
3. Unzip the archive into a folder on your hard drive, e.g. `C:\IRIS_Tbx`. This folder is called the IRIS root.
4. After installing a new version of IRIS, we recommend that you remove all older versions of IRIS from the Matlab search path, and restart Matlab.

Getting started

Each time you want to start working with IRIS, run the following line

```
>> addpath C:\IRIS_Tbx; irisstartup
```

where `C:\IRIS_Tbx` needs to be, obviously, replaced with the proper IRIS root folder chosen during installation.

Alternatively, you can put the IRIS root folder permanently on the Matlab seatch path (using the menu File - Set Path), and only run the `irisstartup` command at the beginning of each IRIS session.

See also the section on Starting and quitting IRIS P9.

Syntax highlighting

You can get the IRIS model files syntax-highlighted. Syntax highlighting improves enormously the readability of the files: it helps you understand the model better, and discover typos and mistakes more quickly.

Add any number of extensions you want to use for model files (such as `'model'` or `'iris'`, there is really no limitation) to the Matlab editor. Open the menu Home - Preferences, unfold Editor/Debugger and choose Language. Make sure Matlab is selected at the top as the language. Use the Add button in the File extensions panel to associate any number of new extensions with the editor. Re-start the editor. The IRIS model files will be syntax highlighted from that moment on.

Third-party components distributed with IRIS

- X13-ARIMA-SEATS (formerly X12-ARIMA, X11-ARIMA). Courtesy of the U.S. Census Bureau, the X13-ARIMA-SEATS (formerly X12-ARIMA) program is now incoporated in, and distributed with IRIS. No extra installation or setup is needed.

2 Starting, quitting, and configuring IRIS

This section describes how to start and quit an IRIS session, and how to customise some of the IRIS configuration options.

The most common way of starting an IRIS session (after you have installed the IRIS files on your disk) is to run the following line in the Matlab command window:

```
addpath C:\IRIS_Tbx; irisstartup();
```

The first command, addpath, adds the IRIS root folder to the Matlab search path. The second command, irisstartup, initialises IRIS and puts the other necessary IRIS subfolders, classes, and internal packages on the search path. Never add these other subfolders, classes and packages to the search path by yourself.

Starting and quitting IRIS

- irisstartup P15 - Start an IRIS session.
- irisfinish P10 - Close the current IRIS session.
- iriscleanup P10 - Remove IRIS from Matlab and clean up.

Getting information about IRIS

- irisget P10 - Query current IRIS config options.
- irisman P12 - Open IRIS Reference Manual PDF.
- irisroot P13 - Current IRIS root folder.
- irisrequired P12 - Throw error if the installed version of IRIS fails to comply with the required minimum.
- irisversion P17 - Current IRIS version.

Changes in configuration

- irisset P14 - Change configurable IRIS options.
- irisreset P13 - Reset IRIS configuration options to start-up values.
- irisuserconfig P16 - User configuration file called at the IRIS start-up.

Getting on-line help on configuration functions

```
help config
help function_name
```

■ iriscleanup

Remove IRIS from Matlab and clean up

Syntax

```
iriscleanup
```

Description

This script removes IRIS folders, including the root folder, from both the Matlab search path, and clears persistent variables in some of the backend functions. A short message is displayed with the list of folders removed from the path.

■ irisfinish

Close the current IRIS session

Syntax

```
irisfinish
irisfinish -shutup
```

Description

This function removes all IRIS subfolders from the temporary Matlab search path, and clears persistent variables in some of the backend functions. A short message is displayed with the list of subfolders removed from the path unless you call use the option `-shutup`. Note that the IRIS root folder stays on the permanent Matlab path.

Example

■ irisget

Query current IRIS config options

Syntax

```
Value = irisget(Option)
S = irisget()
```

Input arguments

- Option [char] - Name of the queried IRIS configuration option.

Output arguments

- Value [...] - Current value of the queried configuration option.
- S [struct] - Structure with all configuration options and their current values.

Description

You can view any of the modifiable options listed in irisset P14, plus the following non-modifiable ones (these cannot be changed by the user):

- 'userConfigPath=' [char] - The path to the user configuration file called by the last executed irisstartup P15.
- 'irisRoot=' [char] - The current IRIS root directory.
- 'version=' [char] - The current IRIS version string.

When called without any input arguments, the irisget function returns a struct with all options and their current values.

When used as input arguments in the irisget function, the option names are case-insensitive. When referring to field names of an output struct returned by the irisget function, all option names are lower-case and case-sensitive.

Example

```
irisget('dateFormat')
ans =
YFP

g = irisget();
```

```
g.dateformat
ans =
YFP
```

---

irisman

Open IRIS Reference Manual PDF

Syntax

```
irisman
```

---

irisrequired

Throw error if the installed version of IRIS fails to comply with the required minimum

Syntax

```
irisrequired(V)
```

Input arguments

- `V` [char] - Text string describing the oldest acceptable distribution of IRIS.

Description

If the version of IRIS present on the computer does not comply with the minimum requirement `v`, an error is thrown.

Example

All of the three calls are valid:

```
irisrequired(20111222);
irisrequired('20111222');
irisrequired 20111222;
```

---

irisreset

Reset IRIS configuration options to start-up values

Syntax

```
irisreset
```

Description

The irisreset function resets all configuration options to their default factory values, or to those in the active irisuserconfig.m file (if one exists).

---

irisroot

Current IRIS root folder

Syntax

```
irisroot
X = irisroot()
```

Output arguments

- X [char] - Path to the IRIS root folder.

Description

The irisroot function is equivalent to the following call to irisget P10

```
irisget('irisroot')
```

irisset

Change configurable IRIS options

Syntax

```
irisset(Option,Value)
irisset(Option,Value,Option,Value,...)
```

Input arguments

- Option [char] - Name of the IRIS configuration option that will be modified.
- Value [...] - New value that will be assigned to the option.

Modifiable IRIS configuration options

Dates and formats

- 'dateFormat=' [char | 'YPF'] - Date format used to display dates in the command window, CSV databases, and reports. Note that the default date format for graphs is controlled by the 'plotdateformat=' option. The default 'YFP' means that the year, frequency letter, and period is displayed. See also help on dat2str P280 for more date formatting details. The 'dateformat=' option is also found in many IRIS functions whenever it is relevant, and can be used to overwrite the 'irisset=' settings.
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months; this option can be used whenever you want to replace the default English names with your local language.
- 'plotDateFormat=' [char | struct('yy','Y','hh','Y:P','qq','Y:P','bb','Y:P','mm','Y:P','ww','Y:P')] - Default date formats used to display dates in graphs including graphs in reports. The default date formats are set individually for each of the 6 ate frequencies in a struct with the following fields: .yy, .hh, .qq, .bb, .mm, .ww. Dates with indeterminate frequency are printed as plain numbers.

- 'tseriesFormat=' [char | empty] - Format string for displaying time series data on the screen. See help on the Matlab sprintf function for how to set up format strings. If empty the default format of the num2str function is used.
- tseriesMaxWSpace=' [numeric | 5] - Maximum number of white spaces printed between individual columns of a multivariate tseries object on the screen.
- 'standinMonth=' [numeric | 'last' | *1*] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.
- 'wwDay=' ['Mon' | 'Tue' | 'Wed' | 'Thu' | 'Fri' | 'Sat' | 'Sun'] - Day of week that will represent weeks in date strings, see dates/dat2str P280, and when weekly tseries objects are displayed on the screen.

External tools used by IRIS

- 'pdflatexPath=' [char] - Location of the pdflatex.exe program. This program is called to compile report and publish m-files. By default, IRIS attempts to locate pdflatex.exe by running TeX's kpsewhich, and which on Unix platforms.
- 'epstopdfPath=' [char] - Location of the epstopdf.exe program. This program is called to convert EPS graphics files to PDFs in reports.

Description

Example

---

irisstartup

Start an IRIS session

Syntax

```
irisstartup
irisstartup -shutup
```

Description

We recommend that you keep the IRIS root directory on the permanent Matlab search path. Each time you wish to start working with IRIS, you run irisstartup form the command line. At the

end of the session, you can run irisfinish P10 to remove IRIS subfolders from the temporary Matlab search path, and to clear persistent variables in some of the backend functions.

The irisstartup P15 performs the following steps:

- Adds necessary IRIS subdirectories to the temporary Matlab search path.
- Removes redundant IRIS folders (e.g. other or older installations) from the Matlab search path.
- Resets IRIS configuration options to default, updates the location of TeX/LaTeX executables, and calls irisuserconfig P16 to modify the configuration option.
- Associates the default IRIS extensions with the Matlab Editor. If they had not been associated before, Matlab must be re-started for the association to take effect.
- Prints an introductory message on the screen unless irisstartup is called with the -shutup input argument.

---

irisuserconfig

User configuration file called at the IRIS start-up

Syntax

```
function c = irisuserconfig(c)
    c.option = value;
    c.option = value;
    ...
end
```

Description

You can create your own configuration file to modify the general IRIS options of your choosing at each IRIS start-up. The file must be saved as irisuserconfig.m on the Matlab search path.

The irisuserconfig.m file must be an m-file function taking one input argument (a struct with the factory settings), and returning one output argument (a struct with the user-modified settings); see irisset P14 for the list of options you can change. In addition, you can also add your own new options, which will be then also accessible through irisset P14 and irisget P10.

Example

If you want the names of months to be displayed in Finnish, create the following m-file and save it in a folder which is on the Matlab search path:

```
function c = irisuserconfig(c)
    c.months = { ...
        'Tammikuu','Helmikuu','Maaliskuu', ...
        'Huhtikuu','Toukokuu','Kesakuu', ...
        'Heinakuu','Elokuu','Syyskuu', ...
        'Lokakuu','Marraskuu','Joulukuu'};
end
```

This modification will take effect after you next run irisstartup P15. Your graphs will be then fluent in Finnish:

```
x = tseries(mm(2009,1):mm(2009,6),@rand);
plot(x,'dateformat','MmmmYY');
```

---

irisversion

Current IRIS version

Syntax

```
irisversion
X = irisversion()
```

Output arguments

- X [char] - String describing the currently installed IRIS version.

Description

The version string is the distribution date in a yyyymmdd format. The irisversion function is equivalent to the call irisget('version').

3 Getting Online Help

Use either `idoc` or `help` to get help on IRIS functions:

- `idoc` displays the help topic in an HTML browser window.
- `help` displays the help topic in the command window,

The following help topics are available:

```
idoc dates
idoc dates/function_name
idoc dbase
idoc dbase/function_name
idoc modellang
idoc modellang/keyword
idoc model
idoc model/function_name
idoc rpteq
idoc rpteq/function_name
idoc plan
idoc plan/function_name
idoc grouping
idoc grouping/function_name
idoc poster
idoc poster/function_name
idoc logdist
idoc logdist/function_name
idoc namedmat
idoc namedmat/function_name
idoc VAR
idoc VAR/function_name
idoc SVAR
idoc SVAR/function_name
idoc BVAR
idoc BVAR/function_name
idoc FAVAR
idoc FAVAR/function_name
idoc dates
idoc dates/function_name
idoc tseries
idoc tseries/function_name
idoc trec
```

```
idoc trec/function_name
idoc dbase
idoc dbase/function_name
idoc report
idoc report/function_name
idoc grfun
idoc grfun/function_name
```

Part II —
Model Development and Operation

4 Model File Language

Model file language is used to write model files. The model files are plain text files (saved under any filename with any extension) that describes the model: its equations, variables, parameters, etc. The model file, on the other hand, does not describe what to do with the model. To run the tasks you want to perform with the model, you need first to load the model file into Matlab using the `model` P122 function. This function creates a model object. Then you write your own m-files using Matlab and IRIS functions to perform the desired tasks with the model object.

Why do all the keywords (except pseudofunctions) start with an exclamation point? Why do the comments have the same style as in Matlab? Why do substitutions and steady-state references use the dollar sign? Because this way, you can get the model files syntax-highlighted in the Matlab editor. Syntax highlighting improves enormously the readability of the files, and helps understand the model more quickly. See the setup instructions P7 for more details.

Variables, parameters, substitutions and functions

- `!transition_variables` P49 - List of transition variables.
- `!transition_shocks` P48 - List of transition shocks.
- `!measurement_variables` P40 - List of measurement variables.
- `!measurement_shocks` P39 - List of measurement shocks.
- `!exogenous_variables` P27 - List of exogenous variables.
- `!parameters` P41 - List of parameters.
- `!autoexogenise` P26 - Definition of variable/shock pairs for use in autoexogenised simulation plans.

Equations

- `!transition_equations` P47 - Block of transition equations.
- `!measurement_equations` P38 - Block of measurement equations.
- `!dtrends` P26 - Block of deterministic trend equations.
- `!links` P36 - Define dynamic links.
- `!sstate_update` P44 - Block of steady-state updating equations.
- `!reporting_equations` P43 - Block of reporting equations.

Linearised and log-linearised variables

- `!log_variables` P37 - List of log-linearised variables.
- `!all_but` P25 - Inverse list of log-linearised variables.
- `<...>` P55 - Regular expression in log variable list.

Special operators

- !! P24 - Steady-state version of an equation.
- !ttrend P50 - Linear time trend in deterministic trend equations.
- {...} P64 - Lag or lead.
- & P53 - Reference to the steady-state level of a variable.
- =# P55 - Mark an equation for exact non-linear simulation.
- '...!!...' P54 - Beginning of aliasing inside descriptions and labels.

Pseudofunctions

Pseudofunctions do not start with an exclamation point.

- min P58 - Define loss function for optimal policy.
- diff P56 - First difference pseudofunction.
- dot P58 - Gross rate of growth pseudofunction.
- difflog P57 - First log-difference pseudofunction.
- movavg P60 - Moving average pseudofunction.
- movgeom P61 - Moving geometric average pseudofunction.
- movprod P62 - Moving product pseudofunction.
- movsum P63 - Moving sum pseudofunction.

Preparser control commands

- !substitutions P44 - Define text substitutions.
- $[...]$ P51 - Pseudosubstitutions.
- !import P35 - Include the content of another model file.
- !export P28 - Create a carry-around file to be saved on the disk.
- !if...!elseif...!else...!end P33 - Choose block of code based on logical condition.
- !switch...!case...!end P45 - Switch among several cases based on expression.
- !for...!do...!end P29 - For loop for automated creation of model code.
- % P52 - Line comments.
- %{...%} P53 - Block comments.

Getting on-line help on model file language

When getting help on model file language, type the names of the keywords and commands without the exclamation point:

```
help modellang
help modellang/keyword
```

```
help modellang/command
help modellang/pseudofunction
```

Matlab functions and user functions in model files

You can use any of the built-in functions (Matlab functions, functions within the Toolboxes you have on your computer, and so on). In addition, you can also use your own functions (written as an m-file) as long as the m-file is on the Matlab search path or in the current directory.

In your own m-file functions, you can also (optionally) supply the first derivatives that will be used to compute Taylor expansions when the model is being solved, and the second derivatives that will be used when the function occurs in a loss function.

When asked for the derivatives, the function is called with two extra input arguments on top of that function's regular input arguments. The first extra input argument is a text string `'diff'` (indicating the call to the function is supposed to return a derivative). The second extra input argument is a number or a vector of two numbers; it determines with respect to which input argument or arguments the first derivative or the second derivative is requested.

For instance, your function takes three input arguments, `myfunc(x,y,z)`. To be able to supply derivates avoiding thus numerical differentiation, the function must be written so that the following three calls

```
myfunc(x,y,z,'diff',1)
myfunc(x,y,z,'diff',2)
myfunc(x,y,z,'diff',3)
```

return the first derivative wrt to the first, second, and third input argument, respectively, while

```
myfunc(x,y,z,'diff',[1,2])
```

returns the second derivative wrt to the first and second input arguments. Note that second derivatives are only needed for functions that occur in an equation defining optimal policy objective, min P58.

If any of these calls fail, the respective derivative will be simply evaluated numerically.

Basic rules IRIS model files

- There can be four types of equations in IRIS models: transition equations which are simply the endogenous dynamic equations, measurement equations which link the model to observables, deterministic trend equations which can be added at the top of measurement equations, and dynamic links which can be used to link some parameters or steady-state values to each other.

- There can be two types of variables and two types of shocks in IRIS models: transition variables and shocks, and measurement variables and shocks.
- Each model must have at least one transition (aka endogenous) variable and one transition equation.
- Each variable, shock, or parameter must be declared in the appropriate declaration section.
- The declaration sections and equations sections can be written in any order.
- You can have as many declaration sections or equations sections of the same kind as you wish in one model file; they all get combined together at the time the model is being loaded.
- Transition variables can occur with lags and leads in transition equations. Transition variables cannot, though, have leads in measurement equations.
- Measurement variables and the shocks cannot have any lags or leads.
- Transition shocks cannot occur in measurement equations, and the measurement shocks cannot occur in transition equations.
- Exogenous variables can only occur in dtrends (deterministic trend equations), and must be always supplied in the input database to commands like `model/simulate`, `model/jforecast`, `model/filter`, `model/estimate`, etc. They are not returned in the output databases.
- You can choose between linearisation and log-linearisation for each individual transition and measurement variable. Shocks are always linearised. Exogenous variables must be always introduced so that their effect on the respective measurement variable is linear.

!!

Steady-state version of an equation

Syntax

```
FullEquation !! SteadyStateEquation;
```

Description

For each transition or measurement equation, you can provide a separate steady-state version of it. The steady-state version is used when you run the functions `sstate` P147 and `chksstate` P79, the latter unless you change the option `'eqtn='`. This is useful when you can substantially simplify

some parts of the full dynamic equations, and help therefore the numerical solver to achieve faster and possibly laso more accurate results.

Why is a double exclamation point, !!, used to start the steady-state versions of equations? Because if you associate your model file extension(s) (such as 'mod' or 'model') with the Matlab editor, anything after an exclamation point is displayed red making it easier to spot the steady-state equations.

Example

The following steady state version will be, of course, valid only in stationary models where we can safely remove lags and leads.

```
Lambda = Lambda{1}*(1+r)*beta !! r = 1/beta - 1;
```

Example

```
log(A) = log(A{-1}) + epsilon_a !! A = 1;
```

■ !all_but

Inverse list of log-linearised variables

Syntax

```
!log_variables
    !all_but
    VariableName, VariableName,
    VariableName, ...
```

Description

See help on !log_variables P37.

!autoexogenise

Definition of variable/shock pairs for use in autoexogenised simulation plans

Syntax

```
!autoexogenise
    VariableName := ShockName; VariableName := ShockName;
    VariableName := ShockName;
```

Description

The section !autoexogenise defines variable/shock pairs that can be used to automate the creation of exogenise-endogenise types of simulation plans P167 using the function autoexogenise P168.

Example

!dtrends

Block of deterministic trend equations

Syntax for linearised measurement variables

```
!dtrends
    VariableName += Expression;
    VariableName += Expression;
    ...
```

Syntax for log-linearised measurement variables

```
!dtrends
    log(VariableName) += Expression;
    log(VariableName) += Expression;
    ...
```

Syntax with equation labels

```
!dtrends
    'Equation label' VariableName += Expression;
    'Equation label' LOG(VariableName) += Expression;
```

Description

Example

```
!dtrends
    Infl += pi_;
    Rate += rho_ + pi_;
```

■ !exogenous_variables

List of exogenous variables

Syntax

```
!exogenous_variables
    VariableName, VariableName, ...
    ...
```

Syntax with descriptors

```
!exogenous_variables
    VariableName, VariableName, ...
    'Description of the variable...' VariableName
```

Syntax with steady-state values

```
!exogenous_variables
    VariableName, VariableName, ...
    VariableName = Value
```

Description

The !exogenous_variables keyword starts a new declaration block for exogenous variables, i.e. variables that can appear only in !dtrends P26 equations. The names of the variables must be separated by commas, semi-colons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each variable must be declared (exactly once).

You can add descriptors to the variables (enclosed in single or double quotes, preceding the name of the variable); these will be stored in, and accessible from, the model object. You can also assign steady-state values to the variables straight in the model file (following an equal sign after the name of the variable); this is, though, rather rare and unnecessary practice because you can assign and change steady-state values more conveniently in the model object.

Example

```
!exogenous_variables
    X, 'Tax effects' Y
    'Population growth effects' Z = 0 + 0.5i;
```

!export

Create a carry-around file to be saved on the disk

Syntax

```
!export(FileName)
    FileContents
!end
```

Description

You can include in the model file the contents of files you need or want to carry around together with the model; a typical example is your own m-file functions used in the model equations.

The file or files are created and save under the name specified in the !export keyword at the time you load the model using the function model P122. The contents of the export files is are also stored in the model objects. You can manually re-create and re-save the files by running the function export P91.

If no filename is provided or FileName is empty, the corresponding !export block is discarded with no error or warning.

■ !for...!do...!end

For loop for automated creation of model code

Abbreviated syntax (cannot be nested)

```
!for
    ListOfTokens
!do
    Template
!end
```

Full syntax

```
!for
    ?ControlName = ListOfTokens
!do
    Template
!end
```

Description

Use the '!for...!do...!end' command to specify a template and let the IRIS preparser automatically create multiple instances of the template by iterating over a list of tokens. The preparser cycles over the individual strings from the list; in each iteration, the current string is used to replace all occurences of the control variable in the template. The name of the control name is either implicitly a question mark, '?', in the abbreviated syntax, or any string starting with a question mark and not containing blank spaces, question marks (other than the leading question mark), colons or periods; for example, '?x', '?#', '?NAME+'.

The tokens (text strings) in the list must be separated by commas, blank spaces, or line breaks and they themselves must not contain any of those. In each iteration,

- all occurrences of the control variable in the template are replaced with the currently processed string;

- all occurrences in the template of ?.ControlName are replaced with the currently processed string converted to lower case; this option is NOT available with the abbreviated syntax;
- all occurrences in the template of ?:ControlName are replaced with the currently processed string converted to upper case; this option is NOT available with the abbreviated syntax;

The list of tokens can be based on Matlab expressions. Use the pseudosubstitution P51 syntax to this end: Enclose an expression in dollar-square brackets, $[...]$. The expression must evaluate to either a numeric vector, a char vector, or a cell array of numerics and/or strings; the value will be then converted to a comma-separted list.

Example

In a model code file, instead of writing a number of definitions of growth rates like the following ones

```
dP = P/P{-1} - 1;
dW = W/W{-1} - 1;
dX = X/X{-1} - 1;
dY = Y/Y{-1} - 1;
```

you can use '!for...!do...!end' as follows:

```
!for
    P, W, X, Y
!do
    d? = ?/?{-1} - 1;
!end
```

Example

We redo the example 1, but using now the fact that you can have as many variable declaration sections or equation sections as you wish. The '!for...!do...!end' structure can therefore not only produce the equations for you, but also make sure all the growth rate variables are properly declared.

```
!for
    P, W, X, Y
!do
    !transition_variables
        d?
```

```
    !transition_equations
        d? = ?/?{-1} - 1;
!end
```

The preparser expands this structure as follows:

```
!transition_variables
    dP
!transition_equations
    dP = P/P{-1} - 1;
!transition_variables
    dW
!transition_equations
    dW = W/W{-1} - 1;
!transition_variables
    dX
!transition_equations
    dX = X/X{-1} - 1;
!transition_variables
    dY
!transition_equations
    dY = Y/Y{-1} - 1;
```

Obviously, you now do not include the growth rate variables in the section where you declare the rest of the variables.

Example

In a model code file, instead of writing a number of autoregression processes like the following ones

```
X = rhox*X{-1} + ex;
Y = rhoy*Y{-1} + ey;
Z = rhoz*Z{-1} + ez;
```

you can use '!for...!do...!end' as follows:

```
!for
    ?# = X, Y, Z
!do
    ?# = rho?.#*?{-1} + e?.#;
!end
```

Example

We redo Example 3, but now for six variables named 'A1', 'A2', 'B1', 'B2', 'C1', 'C2', nesting two '!for...!do...!end' structures one within the other:

```
!for
    ?letter = A, B, C
!do
    !for
        ?number = 1, 2
    !do
        ?letter?number = rho?.letter?number*?letter?number{-1}
            + e?.letter?number;
    !end
!end
```

The preparser produces the following six equations:

```
A1 = rhoa1*A1{-1} + ea1;
A2 = rhoa2*A2{-1} + ea2;
B1 = rhob1*B1{-1} + eb1;
B2 = rhob2*B2{-1} + eb2;
C1 = rhoc1*C1{-1} + ec1;
C2 = rhoc2*C2{-1} + ec2;
```

Example

We use a Matlab expression (the colon operator) to simplify the list of tokens. The following block of code

```
!for
    1, 2, 3, 4, 5, 6, 7
!do
    a? = a?{-1} + res_a?;
!end
```

can be simplified as follow:

```
!for
    $[ 1 : 7 ]$
```

```
!do
    a? = a?{-1} + res_a?;
!end
```

■ !if...!elseif...!else...!end

Choose block of code based on logical condition

Syntax with else and elseif clauses

```
!if Condition1
    Block1
!elseif Condition2
    Block2
!elseif Condition3
...
!else
    Block3
!end
```

Syntax with an else clause only

```
!if Condition1
    Block1
!else
    Block2
!end
```

Syntax without an else clause

```
!if Condition
    Block1
!end
```

Description

The !if...!elseif...!else...!end command works the same way as its counterpart in the Matlab programming language.

Use the !if...!else...!end command to create branches or versions of the model code. Whether a block of code in a particular branch is used or discarded, depends on the condition after the opening !if command and the conditions after subsequent !elseif commands if present. The condition must be a Matlab expression that evaluates to true or false. The condition can refer to model parameters, or to other fields included in the database passed in through the option 'assign=' in the model P122 function.

Example

```
!if B < Inf
    % This is a linearised sticky-price Phillips curve.
    pi = A*pi{-1} + (1-A)*pi{1} + B*log(mu*rmc);
!else
    % This is a flexible-price mark-up rule.
    rmc = 1/mu;
!end
```

If you set the parameter B to Inf in the parameter database when reading in the model file, then the flexible-price equatio, rmc = 0, is used and the Phillips curve equation discarded. To use the Phillips curve equation instead, you need to re-read the model file with B set to a number other than Inf. In this example, B needs to be, obviously, declared as a model parameter.

Example

```
!if exogenous == true
    x = y;
!else
    x = rho*x{-1} + epsilon;
!end
```

When reading the model file in, create a parameter database, include at least a field named exogenous in it, and use the 'assign=' option to pass the database in. Note that you do not need to declare exogenous as a parameter in the model file.

```
P = struct();
P.exogenous = true;
```

```
...
m = model('my.model','assign=',P);
```

In this case, the model will contain the first equation, `x = rho*x{-1} + epsilon;` will be used, and the other discarded. To use the other equation, `x = y`, you need to re-read the model file with exogenous set to false:

```
P = struct();
P.exogenous = false;
...
m = model('my.model','assign=',P);
```

You can also use an abbreviate syntax to assign control parameters when readin the model file; for instance

```
m = model('my.model','exogenous=',true);
```

!import

Include the content of another model file

Syntax

```
!import(FileName)
```

Description

The `!import` command loads the content of the specified file `FileName`. This allows you to split the model code into several parts (each saved in a separate file) and to reuse some bits of the model.

Example

```
!import(mesurement_equations.model)
```

!links

Define dynamic links

Syntax

```
!links
   ParameterName := Expression;
   VariableName := Expression;
```

Syntax with equation labels

```
!links
   'Equation label' ParameterName := Expression;
   'Equation label' VariableName := Expression;
```

Description

The dynamic links relate a particular parameter (or steady-state value) on the LHS to a function of other parameters or steady-state values on the RHS. Expression can be any expression involving parameter names, variables names, Matlab functions and constants, or your own m-file functions on the path; it must not refer to any lags or leads. Expression must evaluate to a single number. It is the user's responsibility to properly handle the imaginary (i.e. growth) part of the steady-state values.

The links are automatically refreshed in solve P143, sstate P147, and chksstate P79 functions, and also in each iteration within the estimate P85 function. They can also be refreshed manually by calling refresh P127.

The links must not involve parameters occuring in !dtrends P26 equations that will be estimated using the 'outoflik=' option of the estimate P85 function.

Example

```
!links
   R := 1/beta;
   alphak := 1 - alphan - alpham;
```

!log_variables

List of log-linearised variables

Syntax

```
!log_variables
    VariableName, VariableName,
    VariableName, ...
```

Syntax with inverted list

```
!log_variables
    !all_but
    VariableName, VariableName,
    VariableName, ...
```

Syntax with regular expression(s)

```
!log_variables
    VariableName, VariableName,
    VariableName, ...
    <REGEXP>, <REGEXP>, ...
```

Description

List all log variables under this headings. Only measurement or transition variables can be declared as log variables.

In non-linear models, all variables are linearised around the steady state or a balanced-growth path. If you wish to log-linearise some of them instead, put them on a `!log_variables` list. You can also use the `!all_but` keyword to indicate an inverse list: all variables will be log-linearised except those listed.

To create the list of log variables, you can also use regular expressions, each enlosed in a pair of angle brackets, < and >. All measurement and transition variables whose names match one of the regular expressions will be declared as log variables. See also help on regular expressions in the Matlab documentation.

Example

The following block of code will cause the variables Y, C, I, and K to be declared as log variables, and hence log-linearised in the model solution, while r and pie will be linearised:

```
!transition_variables
    Y, C, I, K, r, pie

!log_variables
    Y, C, I, K
```

You can do the same job by writing

```
!transition_variables
    Y, C, I, K, r, pie

!log_variables
    !all_but
    r, pie
```

Example

We again achieve the same result as above, but now using a regular expression.

```
!transition_variables
    Y, C, I, K, r, pie

!log_variables
    <[A-Z]\w*>
```

The regular expression [A-Z]\w* selects all variables whose names start with an upper-case letter. Hence, again the variables Y, C, I, and K will be declared as log variables.

■ !measurement_equations

Block of measurement equations

Syntax

```
!measurement_equations
    Equation1;
    Equation2;
    Equation3;
    ...
```

Syntax with equation labels

```
!measurement_equations
    Equation1;
    'Equation label' Equation2;
    Equation3;
    ...
```

Description

The !measurement_equations keyword starts a new block of measurement equations; the equations can stretch over multiple lines and must be separated by semi-colons. You can have as many equation blocks as you wish in any order in your model file: They all get combined together when you read the model file in.

You can add descriptive labels to the equations (in single or double quotes, preceding the equation); these will be stored in, and accessible from, the model object.

Example

```
!measurement_equations
    'Inflation observations' Infl = 40*(P/P{-1} - 1);
```

■ !measurement_shocks

List of measurement shocks

Syntax

```
!measurement_shocks
    ShockName, ShockName, ...
    ...
```

Syntax with descriptors

```
!measurement_shocks
    ShockName, ShockName, ...
    'Description of the shock...' ShockName
```

Description

The !measurement_shocks keyword starts a new declaration block for measurement shocks (i.e. shocks or errors to measurement equation); the names of the shocks must be separated by commas, semicolons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each shock must be declared (exactly once).

You can add descriptors to the shocks (enclosed in single or double quotes, preceding the name of the shock); these will be stored in, and accessible from, the model object.

Example

```
!measurement_shocks
    u1, 'Output measurement error' u2
    u3
```

■ !measurement_variables

List of measurement variables

Syntax

```
!measurement_variables
    VariableName, VariableName, ...
    ...
```

Syntax with descriptors

```
!measurement_variables
    VariableName, VariableName, ...
    'Description of the variable...' VariableName
```

Syntax with steady-state values

```
!measurement_variables
    VariableName, VariableName, ...
    VariableName = Value
```

Description

The !measurement_variables keyword starts a new declaration block for measurement variables (i.e. observables); the names of the variables must be separated by commas, semi-colons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each variable must be declared (exactly once).

You can add descriptors to the variables (enclosed in single or double quotes, preceding the name of the variable); these will be stored in, and accessible from, the model object. You can also assign steady-state values to the variables straight in the model file (following an equal sign after the name of the variable); this is, though, rather rare and unnecessary practice because you can assign and change steady-state values more conveniently in the model object.

For each individual variable in a non-linear model, you can also decide if it is to be linearised or log-linearised by listing its name in the !log_variables P37 section.

Example

```
!measurement_variables
    pie, 'Real output' Y
    'Real exchange rate' Z = 1 + 1.05i;
```

■ !parameters

List of parameters

Syntax

```
!parameters
    ParameterName, ParameterName, ...
    ...
```

Syntax with descriptors

```
!parameters
    ParameterName, ParameterName, ...
    'Description of the parameter...' ParameterName
```

Syntax with steady-state values

```
!parameters
    ParameterName, ParameterName, ...
    ParameterName = value
```

Description

The !parameters keyword starts a new declaration block for parameters; the names of the parameters must be separated by commas, semi-colons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each parameters must be declared (exactly once).

You can add descriptors to the parameters (enclosed in single or double quotes, preceding the name of the parameter); these will be stored in, and accessible from, the model object. You can also assign parameter values straight in the model file (following an equal sign after the name of the parameter); this is, though, rather rare and unnecessary practice because you can assign and change parameter values more conveniently in the model object.

Example

```
!parameters
    alpha, 'Discount factor' beta
    'Labour share' gamma = 0.60
```

!reporting_equations

Block of reporting equations

Syntax

```
!reporting_equations
    LhsName1 = Expression1;
    LhsName2 = Expression2;
    LhsName3 = Expression3;
    ...
```

Syntax with equation labels

```
!reporting_equations
    LhsName1 = Expression1;
    'Equation 2' LhsName2 = Expression2;
    LhsName3 = Expression3;
    ...
```

Description

The !reporting_equations keyword starts a new block of reporting equations; the equations can stretch over multiple lines and must be separated by semi-colons. You can have as many equation blocks as you wish in any order in your model file: They all get combined together when you read the model file in.

You can add descriptive labels to the equations (in single or double quotes, preceding the equation); these will be stored in, and accessible from, the model object.

Although they can be included within a model file and are stored withing a model object, reporting equations are, strictly speaking, not part of the model. They are executed separately from the rest of the model, by calling the function reporting P129.

Example

```
!reporting_equations
    'GDP Growth' g = 100*(Y/Y{-1} - 1);
```

!sstate_update

Block of steady-state updating equations

Syntax

```
!sstate_update
    ParameterName1 = Expression1;
    ParameterName2 = Expression2;
    ParameterName3 = Expression3;
    ...
```

Description

Example

!substitutions

Define text substitutions

Syntax

```
!substitutions
    SubsName := TextString;
    SubsName := TextString;
    ...
```

Description

The !substitutions starts a block with substitution definitions. The definition of each substitution must begin with the name of the substitution, followed by a colon-equal sign, :=, and a text string ended with a semi-colon. The semi-colon is not part of the substitution.

The substitutions can be used in any of the model equations, i.e. in transition equations P47, measurement equations P38, deterministic trend equations P26, and dynamic links P36. Each occurence of the name of a substitution enclosed in dollar signs, i.e. $substitution_name$, in model equations will be replaced with the text string from the substitution's definition.

Substitutions can also refer to other substitutions; make sure, though, that they are not recursive. Also, remember to parenthesise the definitions of the substitutions (or the references to them) in the equations properly so that the resulting mathematical expressions are evaluated properly.

Example

```
!substitution
    a := ((omega1+omega2)/(omega1+omega2+omega3));

!transition_equations
    X = $a$^2*Y + (1-$a$^2)*Z;
```

In this example, we assume that omega1, omega2, and omega3 are declared as parameters. The equation will expand to

```
    X = ((omega1+omega2)/(omega1+omega2+omega3))^2*Y + ...
      (1-((omega1+omega2)/(omega1+omega2+omega3))^2)*Z;
```

Note that if had not used the outermost parentheses in the definition of the substitution, the resulting expression would not have given us what we meant: The square operator would have only applied to the denominator.

■ !switch...!case...!otherwise...!end

Switch among several cases based on expression

Syntax with an otherwise clause

```
!switch Expr
    !case Balue1
        Block1
    !case Balue2
        Block2
    ...
    !otherwise
        OtherwiseBlock
!end
```

Syntax without an otherwise clause

```
!switch Expr
    !case Value1
        Block1
    !case Value2
        Block2
    ...
!end
```

Description

The !switch...!case...!otherwise...!end command works the same way as its counterpart in the Matlab programming language.

Use the !switch...!case...!end command to create a larger number of branches of the model code. Which block of code is actually read in and which blocks are discarded depends on which value in the !case clauses matches the value of the !switch expression. This works exactly as the switch...case...end command in Matlab. The expression after the !switch part of the command must must be a valid Matlab expression, and can refer to the model parameters, or to other fields included in the parameter database passed in when you run the model P122 function; see the option 'assign=' P122.

If the expression fails to be matched by any value in the !case clauses, the branch in the !otherwise clause is used. If it is a !switch command without the !otherwise clause, the whole command is discarded. The Matlab function isequal is used to match the !switch expression with the !case values.

Example

```
!switch policy_regime

    !case 'IT'
        r = rho*r{-1} + (1-rho)*kappa*pie{4} + epsilon;

    !case 'Managed_exchange_rate'
        s = s{-1} + epsilon;

    !case 'Constant_money_growth'
        m-m{-1} = m{-1}-m{-2} + epsilon;

!end
```

When reading the model file in, create a parameter database, include at least a field named policy_regime in it, and use the option 'assign=' to pass the database in. Note that you do not need to declare policy_regime as a parameter in the model file.

```
P = struct();
P.policy_regime = 'Managed_exchange_rate';
...
m = model('my.model','assign',P);
```

In this case, the managed exchange rate policy rule, s = s{-1} + epsilon; is read in and the rest of the !switch command is discarded. To use another branch of the !switch command you need to re-read the model file again with a different value assigned to the policy_regime field of the input database.

!transition_equations

Block of transition equations

Syntax

```
!transition_equations
    Equation1;
    Equation2;
    Equation2;
    ...
```

Abbreviated syntax

```
!equations
    Equation1;
    Equation2;
    Equation3;
    ...
```

Syntax with equation labels

```
!transition_equations
    Equation1;
    'Equation label' Equation2;
    Equation3;
    ...
```

Description

The !transition_equations keyword starts a new block of transition equations (i.e. endogenous equations); the equations can stretch over multiple lines and must be separated by semi-colons. You can have as many equation blocks as you wish in any order in your model file: They all get combined together when you read the model file in.

You can add descriptive labels to the equations (in single or double quotes, preceding the equation); these will be stored in, and accessible from, the model object.

Example

```
!transition_equations
    'Euler equation' C{1}/C = R*beta;
```

!transition_shocks

List of transition shocks

Syntax

```
!transition_shocks
    ShockName, ShockName, ...
    ...
```

Abbreviated syntax

```
!shocks
    ShockName, ShockName, ...
    ...
```

Syntax with descriptors

```
!transition_shocks
    ShockName, ShockName, ...
    'Description of the shock...' ShockName
```

Description

The !transition_shocks keyword starts a new declaration block for transition shocks (i.e. shocks to transition equation); the names of the shocks must be separated by commas, semi-colons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each shock must be declared (exactly once).

You can add descriptors to the shocks (enclosed in single or double quotes, preceding the name of the shock); these will be stored in, and accessible from, the model object.

Example

```
!transition_shocks
    e1, 'Aggregate supply shock' e2
    e3
```

!transition_variables

List of transition variables

Syntax

```
!transition_variables
    VariableName, VariableName, ...
    ...
```

Abbreviated syntax

```
!variables
    VariableName, VariableName, ...
    ...
```

Syntax with descriptors

```
!transition_variables
    VariableName, VariableName, ...
    'Description of the variable...' VariableName
```

Syntax with steady-state values

```
!transition_variables
    VariableName, VariableName, ...
    VariableName = Value
```

Description

The !transition_variables keyword starts a new declaration block for transition variables (i.e. endogenous variables); the names of the variables must be separated by commas, semi-colons, or line breaks. You can have as many declaration blocks as you wish in any order in your model file: They all get combined together when you read the model file in. Each variable must be declared (exactly once).

You can add descriptors to the variables (enclosed in single or double quotes, preceding the name of the variable); these will be stored in, and accessible from, the model object. You can also assign steady-state values to the variables straight in the model file (following an equal sign after the name of the variable); this is, though, rather rare and unnecessary practice because you can assign and change steady-state values more conveniently in the model object.

For each individual variable in a non-linear model, you can also decide if it is to be linearised or log-linearised by listing its name in the !log_variables P37 section.

Example

```
!transition_variables
    pie, 'Real output' Y
    'Real exchange rate' Z = 1 + 1.05i;
```

■ !ttrend

Linear time trend in deterministic trend equations

Syntax

```
!ttrend
```

Description

Example

```
!dtrends
    log(Y) += a*!ttrend;
```

[...]

Pseudosubstitutions

Syntax

```
$[Expr]$
```

Description

The expression `Expr` enclosed within `$[...]$` is evaluated as a Matlab expression, and converted to a character string. The expression may refer to parameters passed into the function `model` P122, or to `!for` P29 loop control variable names. The expression must evaluate to a scalar number, a logical scalar, or character string.

Example

The following line of code

```
pie{$[K]$}
```

which is assumed to be part of a model file named `my.model`, will expand to

```
pie{3}
```

in either of the following two calls to the function model:

```
model('my.model','K=',3);

P = struct();
P.K = 3;
model('my.model','assign=',P);
```

Example

The following !for P29 loop

```
!for
    $[ 2 : 4 ]$
!do
    x? = x$[?-1]${-1};
!end
```

will expand to

```
x2 = x1{-1};
x3 = x2{-1};
x4 = x3{-1};
```

%

Line comments

Syntax

```
% Anything from the percent sign until the end of line is discarded.
```

Description

Example

%{...%}

Block comments

Syntax

```
%{ Anything between
the opening block comment sign
and the closing block comment sign
is discarded %}
```

Description

Unlike in Matlab, the opening and closing block comment signs do not need to stand alone on otherwise blank lines. You can even have block comments contained withing a single line.

Example

```
!transition_equations
    x = rho*x{-1} %{ this is a valid block comment %} + epsilon;
```

&

Reference to the steady-state level of a variable

Syntax

```
&VariableName
$VariableName
&VariableName{K}
$VariableName{K}
```

Description

Use either a `&` or `$` sign in front of a variable name to create a reference to that variable's steady-state level in transition or measurement equations. The two signs, `&` and `$`, are interchangeable. Steady-state references may only be used in nonlinear models.

The steady-state reference can include a time shift (a lag or a lead), K. In that case, the steady-state value will be adjusted for steady-state growth backward or forward accordingly.

The steady-state reference will be replaced

- with the variable itself at the time the model's steady state is being calculated, i.e. when calling the function sstate P147;
- with the actually assigned steady-state value at the time the model is being solved, i.e. when calling the function 'solve' P143'.

Example

```
x = rho*x{-1} + (1-rho)*&x + epsilon_x !! x = 1;
```

■ '...!!...'

Beginning of aliasing inside descriptions and labels

Syntax in descriptions of variables, shocks, and parameters

```
'Description !! Alias' Name
```

Syntax in equations labels

```
'Label !! Alias' Equation;
```

Description

When used in descriptions of variables, shocks, and parameters, or in equation labels, the double exclamation mark starts an alias (but the exlamation marks are not included in it). The alias can be used to specify, for example, a LaTeX code associated with the variable, shock, parameter, or equation. The aliases can be retrieved from the model code by using the appropriate query in the function model/get P100.

Example

```
!transition_variables
    'Output gap !! $\hat y_t$` Y_GAP
```

In the resulting model object, the description of the variables `Y_GAP` will be

```
Output gap
```

while its alias will be

```
$\hat y_t$.
```

■ <...>

Regular expression in log variable list

Syntax

```
!log_variables
    <Regexp>, ...
```

Description

See help on !log_variables P37.

■ =#

Mark an equation for exact non-linear simulation

Syntax

```
LHS =# RHS;
```

Description

Equations that have the equal sign marked with an # can be simulated in an exact non-linear mode.

Why is it the channels sign, #, that is used to mark the equations for exact non-linear simulations? Because if you associate your model file extension with the Matlab editor, the channel signs are displayed red making it easier to spot them.

---

diff

First difference pseudofunction

Syntax

```
diff(Expr)
diff(Expr,K)
```

Description

If the input argument `K` is not specified, this pseudofunction expands to

```
((Expr)-(Expr{-1}))
```

If the input argument `K` is specified, it expands to

```
((Expr)-(Expr{K}))
```

The two derived expressions, `Expr{-1}` and `Expr{K}`, are based on `Expr`, and have all its time subscripts shifted by –1 or by `K` periods, respectively.

Example

These two lines

```
diff(Z)
diff(log(X{1})-log(Y{-1}),-2)
```

will expand to

```
((Z)-(Z{-1}))
((log(X{1})-log(Y{-1}))-(log(X{-1})-log(Y{-3})))
```

■ difflog

First log-difference pseudofunction

Syntax

```
difflog(Expr)
difflog(Expr,K)
```

Description

If the input argument `K` is not specified, this pseudofunction expands to

```
(log(Expr)-log(Expr{-1}))
```

If the input argument `K` is specified, it expands to

```
(log(Expr)-log(Expr{K}))
```

The two derived expressions, `Expr{-1}` and `Expr{K}`, are based on `Expr`, and have all its time subscripts shifted by −1 or by `K` periods, respectively.

Example

The following two lines of code

```
difflog(Z)
difflog(X{1}/Y{-1},-2)
```

will expand to

```
(log(Z)-log(Z{-1}))
(log(X{1}/Y{-1})-log(X{-1}/Y{-3}))
```

---

dot

Gross rate of growth pseudofunction

Syntax

```
dot(Expr)
dot(Expr,K)
```

Description

If the input argument k is not specified, this pseudofunction expands to

```
((Expr)/(Expr{-1}))
```

If the input argument k is specified, it expands to

```
((Expr)/(Expr{k}))
```

The two derived expressions, Expr{-1} and Expr{k}, are based on Expr, and have all its time subscripts shifted by –1 or by k periods, respectively.

Example

The following two lines

```
dot(Z)
dot(X+Y,-2)
```

will expand to

```
((Z)/(Z{-1}))
((X+Y)/(X{-2}+Y{-2}))
```

---

min

Define loss function for optimal policy

Syntax

```
min(Disc) Expr;
```

Syntax for exact non-linear simulations

```
min#(Disc) Expr;
```

Description

The loss function must be types as one of the transition equations. The Disc is a parameter or an expression defining the discount factor (applied to future dates), and the expression Expr defines the loss fuction. The Disc expression must not contain a comma.

If you use the min#(Disc) syntax, all equations created by differentiating the lagrangian w.r.t. individual variables will be earmarked for exact nonlinear simulations provided the respective derivative is nonzero. This only makes sense if the loss function is other than quadratic, and hence its derivatives are nonlinear.

There are two types of optimal policy that can be calculated: time-consistent discretionary policy, and time-inconsistent optimal policy with commitment. Use the option 'optimal=' in the function model P122 at the time of loading the model file to switch between these two types of policy; the option can be either 'discretion' (default) or 'commitment'.

Example

This is a simple model file with a Phillips curve and a quadratic loss function.

```
!transition_variables
    x, pi

!transition_shocks
    u

!parameters
    alpha, beta, gamma

!transition_equations
    min(beta) pi^2 + lambda*x^2;
    pi = alpha*pi{-1} + (1-alpha)*pi{1} + gamma*y + u;
```

movavg

Moving average pseudofunction

Syntax

```
movavg(Expr)
movavg(Expr,K)
```

Description

If the second input argument, K, is negative, this function expands to the moving average of the last K periods (including the current period), i.e.

```
(((Expr)+(Expr{-1})+ ... +(Expr{-(K-1)})/-K)
```

where Expr{-N} derives from Expr and has all its time subscripts shifted by -N (if specified).

If the second input argument, K, is positive, this function expands to the moving average of the next K periods ahead (including the current period), i.e.

```
(((Expr)+(Expr{1})+ ... +(Expr{K-1})/K)
```

If the second input argument, K, is not specified, the default value -4 is used (based on the fact that most of the macroeconomic models are quarterly).

Example

The following three lines

```
movavg(Z)
movavg(Z,-3)
movavg(X+Y{-1},2)
```

will expand to

```
(((Z)+(Z{-1})+(Z{-2})+(Z{-3}))/4)
(((Z)+(Z{-1})+(Z{-2}))/3)
(((X+Y{-1})+(X{1}+Y))/2)
```

---

movavg

Moving geometric average pseudofunction

Syntax

```
movgeom(Expr)
movgeom(Expr,K)
```

Description

If the second input argument, K, is negative, this function expands to the moving geometric average of the last K periods (including the current period), i.e.

```
(((Expr)*(Expr{-1})* ... *(Expr{-(K-1)})^(1/-K))
```

where Expr{-N} derives from Expr and has all its time subscripts shifted by -N (if specified).

If the second input argument, K, is positive, this function expands to the moving geometric average of the next K periods ahead (including the current period), i.e.

```
(((Expr)*(Expr{1})* ... *(Expr{K-1})^(1/K))
```

If the second input argument, K, is not specified, the default value -4 is used (based on the fact that most of the macroeconomic models are quarterly).

Example

The following three lines

```
movgeom(Z)
movgeom(Z,-3)
movgeom(X+Y{-1},2)
```

will expand to

```
(((Z)*(Z{-1})*(Z{-2})*(Z{-3}))^(1/4))
(((Z)*(Z{-1})*(Z{-2}))^(1/3))
(((X+Y{-1})*(X{1}+Y))^(1/2))
```

---

movsum

Moving product pseudofunction

Syntax

```
movprod(Expr)
movprod(Expr,K)
```

Description

If the second input argument, K, is negative, this function expands to the moving product of the last K periods (including the current period), i.e.

```
((Expr)*(Expr{-1})* ... *(Expr{-(K-1)})
```

where Expr{-N} derives from Expr and has all its time subscripts shifted by -N (if specified).

If the second input argument, K, is positive, this function expands to the moving product of the next K periods ahead (including the current period), i.e.

```
((Expr)*(Expr{1})* ... *(Expr{K-1})
```

If the second input argument, K, is not specified, the default value -4 is used (based on the fact that most of the macroeconomic models are quarterly).

Example

The following three lines

```
movprod(Z)
movprod(Z,-3)
movprod(X+Y{-1},2)
```

will expand to

```
((Z)*(Z{-1})*(Z{-2})*(Z{-3}))
((Z)*(Z{-1})*(Z{-2}))
((X+Y{-1})*(X{1}+Y))
```

---

movsum

Moving sum pseudofunction

Syntax

```
movsum(Expr)
movsum(Expr,K)
```

Description

If the second input argument, K, is negative, this function expands to the moving sum of the last K periods (including the current period), i.e.

```
((Expr)+(Expr{-1})+ ... +(Expr{-(K-1)})
```

where Expr{-N} derives from Expr and has all its time subscripts shifted by -N (if specified).

If the second input argument, K, is positive, this function expands to the moving sum of the next K periods ahead (including the current period), i.e.

```
((Expr)+(Expr{1})+ ... +(Expr{K-1})
```

If the second input argument, K, is not specified, the default value -4 is used (based on the fact that most of the macroeconomic models are quarterly).

Example

The following three lines

```
movsum(Z)
movsum(Z,-3)
movsum(X+Y{-1},2)
```

will expand to

```
((Z)+(Z{-1})+(Z{-2})+(Z{-3}))
((Z)+(Z{-1})+(Z{-2}))
((X+Y{-1})+(X{1}+Y))
```

■ {...}

Lag or lead

Syntax

```
VariableName{-lag}
VariableName{lead}
VariableName{+lead}
```

Description

To create a lag or a lead of a variable, use a pair of curly brackets.

Example

```
!transition_equations
    x = rho*x{-1} + epsilon_x;
    pi = 1/2*pie{-1} + 1/2*pie{1} + gamma*y + epsilon_pi;
```

5 Models (model Objects)

Model objects are created by loading a model file P21. Once a model object exists, you can use model functions and standard Matlab functions to write your own m-files to perform the desired tasks, such calibrate or estimate the model, find its steady state, solve and simulate it, produce forecasts, analyse its properties, and so on.

Model methods:

Constructor

- model P122 - Create new model object based on model file.

Getting information about model

- addparam P70 - Add model parameters to a database (struct).
- autocaption P73 - Create captions for graphs of model variables or parameters.
- autoexogenise P74 - Get or set variable/shock pairs for use in autoexogenised simulation plans.
- chkredundant P78 - Check for redundant shocks and/or parameters.
- comment P80 - Get or set user comments in an IRIS object.
- eig P84 - Eigenvalues of the transition matrix.
- findeqtn P97 - Find equations by the labels.
- findname P97 - Find names of variables, shocks, or parameters by their descriptors.
- get P100 - Query model object properties.
- iscompatible P108 - True if two models can occur together on the LHS and RHS in an assignment.
- islinear P109 - True for models declared as linear.
- islog P111 - True for log-linearised variables.
- ismissing P111 - True if some initical conditions are missing from input database.
- islocked P110 - Get lock status of dynamic links or sstate update equations.
- isnan P113 - Check for NaNs in model object.
- isname P112 - True for valid names of variables, parameters, or shocks in model object.
- issolved P113 - True if model solution exists.
- isstationary P114 - True if model or specified combination of variables is stationary.
- length P116 - Number of alternative parameterisations.
- omega P126 - Get or set the covariance matrix of shocks.
- sspace P145 - State-space matrices describing the model solution.
- system P153 - System matrices for unsolved model.
- userdata P157 - Get or set user data in an IRIS object.

Referencing model objects

- subsasgn P151 - Subscripted assignment for model and systemfit objects.
- subsref P152 - Subscripted reference for model and systemfit objects.

Changing model objects

- alter P71 - Expand or reduce number of alternative parameterisations.
- assign P71 - Assign parameters, steady states, std deviations or cross-correlations.
- export P91 - Save export files to disk.
- horzcat P106 - Combine two compatible model objects in one object with multiple parameterisations.
- lock P118 - Lock (disable) dynamic links or sstate update equations temporarily.
- refresh P127 - Refresh dynamic links.
- reset P131 - Reset specific values within model object.
- stdscale P150 - Rescale all std deviations by the same factor.
- set P134 - Change modifiable model object property.
- single P142 - Convert solution matrices to single precision.
- unlock P156 - Unlock (enable) locked dynamic links or sstate update equations.

Steady state

- blazer P75 - Reorder steady-state equations into block-recursive structure.
- chksstate P79 - Check if equations hold for currently assigned steady-state values.
- sstate P147 - Compute steady state or balance-growth path of the model.

Solution, simulation and forecasting

- chkmissing P77 - Check for missing initial values in simulation database.
- diffsrf P83 - Differentiate shock response functions w.r.t. specified parameters.
- expand P90 - Compute forward expansion of model solution for anticipated shocks.
- jforecast P115 - Forecast with judgmental adjustments (conditional forecasts).
- icrf P106 - Initial-condition response functions.
- lhsmrhs P117 - Evaluate the discrepancy between the LHS and RHS for each model equation and given data.
- resample P130 - Resample from the model implied distribution.
- reporting P129 - Evaluate reporting equations from within model object.
- shockplot P136 - Short-cut for running and plotting plain shock simulation.
- simulate P137 - Simulate model.
- solve P143 - Calculate first-order accurate solution of the model.
- srf P144 - Shock response functions, first-order solution only.
- tolerance P155 - Get or set model-specific tolerance levels.

Model data

- data4lhsmrhs P81 - Prepare data array for running lhsmrhs.
- emptydb P84 - Create model-specific database with empty tseries for all variables, shocks and parameters.
- rollback P132 - Prepare database for a rollback run of Kalman filter.
- shockdb P135 - Create model-specific database with random shocks.
- sstatedb P149 - Create model-specific steady-state or balanced-growth-path database.
- templatedb P155 - Create model-specific template database.
- zerodb P160 - Create model-specific zero-deviation database.

Stochastic properties

- acf P68 - Autocovariance and autocorrelation function for model variables.
- ifrf P107 - Frequency response function to shocks.
- fevd P91 - Forecast error variance decomposition for model variables.
- ffrf P92 - Filter frequency response function of transition variables to measurement variables.
- fmse P99 - Forecast mean square error matrices.
- vma P158 - Vector moving average representation of the model.
- xsf P159 - Power spectrum and spectral density of model variables.

Identification, estimation and filtering

- bn P76 - Beveridge-Nelson trends.
- diffloglik P82 - Approximate gradient and hessian of log-likelihood function.
- estimate P85 - Estimate model parameters by optimising selected objective function.
- evalsystempriors P90 - Evaluate minus log of system prior density.
- filter P93 - Kalman smoother and estimator of out-of-likelihood parameters.
- fisher P98 - Approximate Fisher information matrix in frequency domain.
- lognormal P122 - Characteristics of log-normal distributions returned from filter of forecast.
- loglik P119 - Evaluate minus the log-likelihood function in time or frequency domain.
- neighbourhood P125 - Evaluate the local behaviour of the objective function around the estimated parameter values.
- regress P128 - Centred population regression for selected model variables.
- VAR P158 - Population VAR for selected model variables.

Getting on-line help on model functions

```
help model
help model/function_name
```

Reference page for model

---

acf

Autocovariance and autocorrelation function for model variables

Syntax

```
[C,R,List] = acf(M,...)
```

Input arguments

- `M` [model] - Solved model object for which the ACF will be computed.

Output arguments

- `C` [namedmat | numeric] - Auto/cross-covariance matrices.
- `R` [namedmat | numeric] - Auto/cross-correlation matrices.
- `List` [cellstr] - List of variables in rows and columns of `C` and `R`.

Options

- `'applyTo='` [cellstr | char | `@all`] - List of variables to which the `'filter='` will be applied; `@all` means all variables.
- `'contributions='` [`true` | `false`] - If `true` the contributions of individual shocks to ACFs will be computed and stored in the 5th dimension of the `C` and `R` matrices.
- `'filter='` [char | empty] - Linear filter that is applied to variables specified by 'applyto'.
- `'nFreq='` [numeric | `256`] - Number of equally spaced frequencies over which the filter in the option `'filter='` is numerically integrated.
- `'order='` [numeric | `0`] - Order up to which ACF will be computed.
- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrices `C` and `R` as either namedmat P204 objects (i.e. matrices with named rows and columns) or plain numeric arrays.
- `'select='` [`@all` | char | cellstr] - Return ACF for selected variables only; `@all` means all variables.

Description

C and R are both N-by-N-by-(P+1)-by-NAlt matrices, where N is the number of measurement and transition variables (including auxiliary lags and leads in the state space vector), P is the order up to which the ACF is computed (controlled by the option 'order='), and NAlt is the number of alternative parameterisations in the input model object, M.

If 'contributions=' true, the size of the two matrices is N-by-N-by-(P+1)-by-E-by-NAlt, where E is the number of all shocks (measurement and transition combined) in the model.

ACF with linear filters

You can use the option 'filter=' to get the ACF for variables as though they were filtered through a linear filter. You can specify the filter in both the time domain (such as first-difference filter, or Hodrick-Prescott) and the frequncy domain (such as a band of certain frequncies or periodicities). The filter is a text string in which you can use the following references:

- 'L', the lag operator, which will be replaced with exp(-1i*freq);
- 'per', the periodicity;
- 'freq', the frequency.

Example

A first-difference filter (i.e. computes the ACF for the first differences of the respective variables):

```
[C,R] = acf(m,'filter=','1-L')
```

Example

The cyclical component of the Hodrick-Prescott filter with the smoothing parameter, *lambda*, 1,600. The formula for the filter follows from the classical Wiener-Kolmogorov signal extraction theory,

$$w(L) = \frac{\lambda}{\lambda + \frac{1}{|(1-L)(1-L)|^2}}$$

```
[C,R] = acf(m,'filter','1600/(1600 + 1/abs((1-L)^2)^2)')
```

Example

A band-pass filter with user-specified lower and upper bands. The band-pass filters can be defined either in frequencies or periodicities; the latter is usually more convenient. The following is a filter which retains periodicities between 4 and 40 periods (this would be between 1 and 10 years in a quarterly model),

```
[C,R] = acf(m,'filter','per >= 4 & per <= 40')
```

---

addparam

Add model parameters to a database (struct)

Syntax

```
D = addparam(M,D)
```

Input arguments

- M [model] - Model object whose parameters will be added to database (struct) D.
- D [struct] - Database to which the model parameters will be added.

Output arguments

- 'D [struct] - Database with the model parameters added.

Description

If there are database entries in D whose names conincide with the model parameters, they will be overwritten.

Example

```
D = struct();
D = addparam(M,D);
```

---

alter

Expand or reduce number of alternative parameterisations

Syntax

```
M = alter(M,N)
```

Input arguments

- `M` [model] - Model object in which the number of paremeterisations will be changed.
- `N` [numeric] - New number of parameterisations.

Output arguments

- `M` [model] - Model object with the new number of parameterisations.

Description

Example

---

assign

Assign parameters, steady states, std deviations or cross-correlations

Syntax

```
[M,Assigned] = assign(M,P)
[M,Assigned] = assign(M,N)
[M,Assigned] = assign(M,Name,Value,Name,Value,...)
[M,Assigned] = assign(M,List,Values)
```

Syntax for fast assign

```
% Initialise
assign(M,List);

% Fast assign
M = assign(M,Values);
...
M = assign(M,Values);
...
```

Syntax for assigning only steady-state levels

```
M = assign(M,'-level',...)
```

Syntax for assignin only steady-state growth rates

```
M = assign(M,'-growth',...)
```

Input arguments

- `M` [model] - Model object.
- `P` [struct] - Database whose fields refer to parameter names, variable names, std deviations, or cross-correlations.
- `N` [model] - Another model object from which all parameteres (including std erros and cross-correlation coefficients), and steady-states values will be assigned that match the name and type in `M`.
- `Name` [char] - A parameter name, variable name, std deviation, cross-correlation, or a regular expression that will be matched against model names.
- `Value` [numeric] - A value (or a vector of values in case of multiple parameterisations) that will be assigned.
- `List` [cellstr] - A list of parameter names, variable names, std deviations, or cross-correlations.
- `Values` [numeric] - A vector of values.

Output arguments

- M [model] - Model object with newly assigned parameters and/or steady states.
- Assigned [cellstr | Inf] - List of actually assigned parameter names, variables names (steady states), std deviations, and cross-correlations; Inf indicates that all values has been assigned from another model object.

Description

Calls with Name-Value or List-Value pairs throw an error if some names in the list are not valid names in the model object. Calls with a database, P, or another model object, N, do not perform this check.

Example

■ autocaption

Create captions for graphs of model variables or parameters

Syntax

```
C = autocaption(M,X,Template,...)
```

Input arguments

- M [model] - Model object.
- X [cellstr | struct | poster] - A cell array of model names, a struct with model names, or a poster P188 object.
- Template [char] - Prescription for how to create the caption; see Description for details.

Output arguments

- C [cellstr] - Cell array of captions, with one for each model name (variable, shock, parameter) found in X, in order of their appearance in X.

Options

- 'corr=' [char | 'Corr $shock1$ X $shock2$'] - Template to create $descript$ and $alias$ for correlation coefficients based on $descript$ and $alias$ of the underlying shocks.
- 'std=' [char | 'Std $shock$'] - Template to create $descript$ and $alias$ for std deviation based on $descript$ and $alias$ of the underlying shock.

Description

The function autocaption can be used to supply user-created captions to title graphs in grfun/plotpp, grfun/plotneigh, model/shockplot, and dbase/dbplot, through their option 'caption='.

The Template can contain the following substitution strings:

- $name$ – will be replaced with the name of the respective variable, shock, or parameter;
- $descript$ – will be replaced with the description of the respective variable, shock, or parameter;
- $alias$ – will be replaced with the alias of the respective variable, shock, or parameter.

The options 'corr=' and 'std=' will be used to create $descript$ and '*alias* for std deviations and cross-correlations of shocks (which cannot be created in the model code). The options are expected to use the following substitution strings:

- '$shock$' – will be replaced with the description or alias of the underlying shock in a std deviation;
- '$shock1$' – will be replaced with the description or alias of the first underlying shock in a cross correlation;
- '$shock2$' – will be replaced with the description or alias of the second underlying shock in a cross correlation.

Example

■ autoexogenise

Get or set variable/shock pairs for use in autoexogenised simulation plans

Syntax fo getting autoexogenised variable/shock pairs

```
A = autoexogenise(M)
```

Syntax fo setting autoexogenised variable/shock pairs

```
M = autoexogenise(M,A)
```

Input arguments

- `M` [model] - Model object.
- `A` [struct | empty] - Database with each field representing a variable/shock pair, A.Variable_Name = 'Shock_Name', that can be used in building simulation plans P167 by the plan function autoexogenise P168.

Output arguments

- `M` [model] - Model object with updated definitions of autoexogenised variable/shock pairs.

Description

Whenever you set the autoexogenised variable/shock pairs, the previously assigned pairs are removed, and replaced with the new ones in `A`. In other words, the new pairs are not added to the existing ones, the replace them.

Example

---

blazer

Reorder steady-state equations into block-recursive structure

Syntax

```
[NameBlk,EqtnBlk] = blazer(M,...)
```

Input arguments

- `M` [model] - Model object.

Output arguments

- `M` [model] - Model object with variables and steady-state equations regrouped to create block-recursive structure.
- `NameBlk` [cell] - Cell of cellstr with variable names in each block.
- `EqtnBlk` [cell] - Cell of cellstr with equations in each block.

Options

- `'endogenize='` [cellstr | char] - List of parameters that will be endogenized in steady equations.
- `'exogenize='` [cellstr | char] - List of transition or measurement variables that will be exogenized in steady equations.

Description

The reordering algorithm first identifies equations with a single variable in each, and variables occurring in a single equation each, and then uses a combination of column and row approximate minimum degree permutations (`colamd`) followed by a Dulmage-Mendelsohn permutation (`dmperm`).

The output arguments `NameBlk` and `EqtnBlk` are 1-by-N cell arrays, where N is the number of blocks, and each cell is a 1-by-Kn cell array of strings, where Kn is the number of variables and equations in block N.

Example

---

bn

Beveridge-Nelson trends

Syntax

```
Outp = bn(M,Inp,Range,...)
```

Input arguments

- M [model] - Solved model object.
- Inp [struct | cell] - Input data on which the BN trends will be computed.
- Range [numeric | char] - Date range on which the BN trends will be computed.

Output arguments

- Outp [struct | cell] - Output data with the BN trends.

Options

- 'deviations=' [true | false] - Input and output data are deviations from balanced-growth paths.
- 'dtrends=' [@auto | true | false] - Measurement variables in input and output data include deterministic trends specified in !dtrends P26 equations.

Description

The BN decomposition is accurate only if the input data have been generated using unanticipated shocks.

Example

---

chkmissing

Check for missing initial values in simulation database

Syntax

```
[Ok,Miss] = chkmissing(M,D,Start)
```

Input arguments

- `M` [model] - Model object.
- `D` [struct] - Input database for the simulation.
- `Start` [numeric] - Start date for the simulation.

Output arguments

- `Ok` [`true` | `false`] - True if the input database `D` contains all required initial values for simulating model `M` from date `Start`.
- `Miss` [cellstr] - List of missing initial values.

Options

- `'error='` [`true` | `false`] - Throw an error if one or more initial values are missing.

Description

This function does not perform any simulation; it only checks for missing initial values in an input database.

Example

---

chkredundant

Check for redundant shocks and/or parameters

Syntax

```
[RedShocks,RedParams] = chkredundant(M)
```

Input arguments

- `M` [model] - Model object.

Output arguments

- RedShocks [cellstr] - List of shocks that do not occur in any model equation.
- RedParams [cellstr] - List of parameters that do not occur in any model equation.

Options

- 'warning=' [true | false] - Throw a warning listing redundant shocks and parameters.
- 'chkShocks=' [true | false] - Check for redundant shocks.
- 'chkParams=' [true | false] - Check for redundant parameters.

Description

Example

■ chksstate

Check if equations hold for currently assigned steady-state values

Syntax

```
[Flag,List] = chksstate(M,...)
[Flag,Discr,List] = chksstate(M,...)
```

Input arguments

- M [model] - Model object.

Output arguments

- Flag [true | false] - True if discrepancy between LHS and RHS is smaller than tolerance level in each equation.
- Discr [numeric] - Discrepancies between LHS and RHS evaluated for each equation at two consecutive times, and returned as two column vectors.
- List [cellstr] - List of equations in which the discrepancy between LHS and RHS is greater than 'tolerance='.

Options

- 'error=' [true | false] - Throw an error if one or more equations fail to hold up to tolerance level.
- 'eqtn=' ['full' | 'sstate'] - Evaluate either full or steady-state equations on steady-state values.
- 'warning=' [true | false] - Display warnings produced by this function.

Description

Example

---

comment

Get or set user comments in an IRIS object

Syntax for getting user comments

```
Cmt = comment(Obj)
```

Syntax for assigning user comments

```
Obj = comment(Obj,Cmt)
```

Input arguments

- Obj [model | tseries | VAR | SVAR | FAVAR | sstate] - One of the IRIS objects.
- Cmt [char] - User comment that will be attached to the object.

Output arguments

- Cmt [char] - User comment that are currently attached to the object.

Description

Example

/bin/bash: lhsmrhs: command not found

---

data4lhsmrhs

Prepare data array for running

Syntax

```
[YXE,List,XRange] = data4lhsmrhs(M,Inp,Range)
```

Input arguments

- `M` [model] - Model object whose equations will be later evaluated by calling `lhsmrhs` P117.
- `Inp` [struct] - Input database with observations on measurement variables, transition variables, and shocks on which `lhsmrhs` P117 will be evaluated.
- `Range` [numeric | char] - Date range on which `lhsmrhs` P117 will be evaluated.

Output arguments

- `YXE` [numeric] - Numeric array with the observations on measurement variables, transition variables, and shocks organised row-wise.
- `List` [cellstr] - List of measurement variables, transition variables and shocks in order of their appearance in the rows of `YXE`.
- `XRange` [numeric] - Extended range including pre-sample and post-sample observations needed to evaluate lags and leads of transition variables.

Description

The resulting array, `YXE`, is `nVar` by `nXPer` by `nData`, where `nVar` is the total number of measurement variables, transition variables, and shocks, `nXPer` is the number of periods including the pre-sample and post-sample periods needed to evaluate lags and leads, and `nData` is the number of alternative data sets (i.e. the number of columns in each input time series) in the input database, `Inp`.

Example

```
YXE = data4lhsmrhs(M,d,range);
D = lhsmrhs(M,YXE);
```

---

diffloglik

Approximate gradient and hessian of log-likelihood function

Syntax

```
[MinusLogLik,Grad,Hess,V] = diffloglik(M,Inp,Range,PList,...)
```

Input arguments

- `M` [model] - Model object whose likelihood function will be differentiated.
- `Inp` [cell | struct] - Input data from which measurement variables will be taken.
- `Range` [numeric | char] - Date range on which the likelihood function will be evaluated.
- `PList` [cellstr] - List of model parameters with respect to which the likelihood function will be differentiated.

Output arguments

- `MinusLogLik` [numeric] - Value of minus the likelihood function at the input data.
- `Grad` [numeric] - Gradient (or score) vector.
- `Hess` [numeric] - Hessian (or information) matrix.
- `V` [numeric] - Estimated variance scale factor if the `'relative='` options is true; otherwise `v` is 1.

Options

- `'chkSstate='` [`true` | `false` | cell] - Check steady state in each iteration; works only in non-linear models.

- 'solve=' [true | false | cellstr] - Re-compute solution for each parameter change; you can specify a cell array with options for the solve function.
- 'sstate=' [true | false | cell] - Re-compute steady state in each differentiation step; if the model is non-linear, you can pass in a cell array with options used in the sstate function.

See help on model/filter P93 for other options available.

Description

Example

---

diffsrf

Differentiate shock response functions w.r.t. specified parameters

Syntax

```
S = diffsrf(M,Range,PList,...)
S = diffsrf(M,NPer,PList,...)
```

Input arguments

- M [model] - Model object whose response functions will be simulated and differentiated.
- Range [numeric | char] - Simulation date range with the first date being the shock date.
- NPer [numeric] - Number of simulation periods.
- PList [char | cellstr] - List of parameters w.r.t. which the shock response functions will be differentiated.

Output arguments

- S [struct] - Database with shock reponse derivatives stowed in multivariate time series.

Options

See model/srf P144 for options available.

Description

Example

---

eig

Eigenvalues of the transition matrix

Syntax

```
e = eig(m)
```

Input arguments

- m [model] - Model object whose eigenvalues will be returned.

Output arguments

- e [numeric] - Array of all eigenvalues associated with the model, i.e. all stable, unit, and unstable roots are included.

Description

Example

---

emptydb

Create model-specific database with empty tseries for all variables, shocks and parameters

Syntax

```
D = emptydb(M)
```

Input arguments

- M [model] - Model for which the empty database will be created.

Output arguments

- D [struct] - Database with an empty tseries object for each variable and each shock, and a vector of currently assigned values for each parameter.

Description

Example

■ estimate

Estimate model parameters by optimising selected objective function

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[PEst,Pos,Cov,Hess,M,V,Delta,PDelta] = estimate(M,D,Range,Est,~Spr,...)
```

Input arguments

- M [model] - Model object with single parameterization.
- D [struct | cell] - Input database or datapack from which the measurement variables will be taken.
- Range [struct | char] - Date range on which the data likelihood will be evaluated.
- Est [struct] - Database with the list of paremeters that will be estimated, and the parameter prior specifications (see below).
- ~SPr [systempriors | empty] - System priors object, systempriors P183; may be omitted.

Output arguments

- PEst [struct] - Database with point estimates of requested parameters.
- Pos [poster] - Posterior, poster P188, object; this object also gives you access to the value of the objective function at optimum or at any point in the parameter space, see the poster/eval P191 function.
- Cov [numeric] - Approximate covariance matrix for the estimates of parameters with slack bounds based on the asymptotic Fisher information matrix (not on the Hessian returned from the optimization routine).
- Hess [cell] - Hess{1} is the total hessian of the objective function; Hess{2} is the contributions of the priors to the hessian.
- M [model] - Model object solved with the estimated parameters (including out-of-likelihood parameters and common variance factor).

The remaining three output arguments, V, Delta, PDelta, are the same as the model/loglik P119 output arguments of the same names.

Options

- 'chkSstate=' [true | false | cell] - Check steady state in each iteration; works only in non-linear models.
- 'evalFrfPriors=' [true | false] - In each iteration, evaluate frequency response function prior density, and include it to the overall objective function to be optimised.
- 'evalLik=' [true | false] - In each iteration, evaluate likelihood (or another data based criterion), and include it to the overall objective function to be optimised.
- 'evalPPriors=' [true | false] - In each iteration, evaluate parameter prior density, and include it to the overall objective function to be optimised.
- 'evalSPriors=' [true | false] - In each iteration, evaluate system prior density, and include it to the overall objective function to be optimised.
- 'filter=' [cell | empty] - Cell array of options that will be passed on to the Kalman filter including the type of objective function; see help on model/filter P93 for the options available.
- 'initVal=' [model | struct | struct] - If struct use the values in the input struct Est to start the iteration; if model use the currently assigned parameter values in the input model, M.

- 'maxIter=' [numeric | 500] - Maximum number of iterations allowed.
- 'maxFunEvals=' [numeric | 2000] - Maximum number of objective function calls allowed.
- 'noSolution=' ['error' | 'penalty' | numeric] - Specifies what happens if solution or steady state fails to solve in an iteration: 'error=' stops the execution with an error message, 'penalty=' returns an extreme value, 1e10, back into the minimization routine; or a user-supplied penalty can be specified as a numeric scalar greater than 1e10.
- 'optimSet=' [cell | empty] - Cell array used to create the Optimization Toolbox options structure; works only with the option 'optimiser=' 'default'.
- 'solve=' [true | false | cellstr] - Re-compute solution in each iteration; you can specify a cell array with options for the solve function.
- 'optimiser=' ['default' | 'pso' | cell | function_handle] - Minimiz ation procedure.
  - 'default': The Optimization Toolbox function fminunc or fmincon will be called depending on the presence or absence of lower and/or upper bounds.
  - 'alps': The age layer population structure evolutionary algorithm will be used. See irisoptim.alps help for more information.
  - 'pso': The particle swarm optimizer will be called. See the irisoptim.pso help for more information.
  - function_handle or cell: Enter a function handle to your own optimization procedure, or a cell array with a function handle and additional input arguments (see below).
- 'sstate=' [true | false | cell | function_handle] - Re-compute steady state in each iteration; you can specify a cell array with options for the sstate function, or a function handle whose behaviour is described below.
- 'tolFun=' [numeric | 1e-6] - Termination tolerance on the objective function.
- 'tolX=' [numeric | 1e-6] - Termination tolerance on the estimated parameters.

Description

The parameters that are to be estimated are specified in the input parameter estimation database, E in which you can provide the following specifications for each parameter:

```
E.parameter_name = { start, lower, upper, logpriorFunc };
```

where start is the value from which the numerical optimization will start, lower is the lower bound, upper is the upper bound, and logpriorFunc is a function handle expected to return the log of the

prior density. You can use the logdist P196 package to create function handles for some of the basic prior distributions.

You can use NaN for start if you wish to use the value currently assigned in the model object. You can use -Inf and Inf for the bounds, or leave the bounds empty or not specify them at all. You can leave the prior distribution empty or not specify it at all.

Estimating nonlinear models

By default, only the first-order solution, but not the steady state is updated (recomputed) in each iteration before the likelihood is evaluated. This behavior is controled by two options, 'solve=' (true by default) and 'sstate=' (false by default). If some of the estimated parameters do affect the steady state of the model, the option 'sstate=' needs to be set to true or to a cell array with steady-state options, as in the function sstate P147, otherwise the results will be grosslly inaccurate or a valid first-order solution will be impossible to find.

When steady state is recomputed in each iteration, you may also want to use the option 'chksstate=' to require that a steady-state check for all model equations be performed.

User-supplied optimization (minimization) routine

You can supply a function handle to your own minimization routine through the option 'optimiser='. This routine will be used instead of the Optim Tbx's fminunc or fmincon functions. The user-supplied function is expected to take at least five input arguments and return three output arguments:

```
[PEst,ObjEst,Hess] = yourminfunc(F,P0,PLow,PHigh,OptimSet)
```

with the following input arguments:

- F is a function handle to the function minimised;
- P0 is a 1-by-N vector of initial parameter values;
- PLow is a 1-by-N vector of lower bounds (with -Inf indicating no lower bound);
- PHigh is a 1-by-N vector of upper bounds (with Inf indicating no upper bounds);
- OptimSet is a cell array with name-value pairs entered by the user through the option 'optimSet='. This option can be used to modify various settings related to the optimization routine, such as tolerance, number of iterations, etc. Of course, you may simply ignore it and leave this input argument unused;

and the following output arguments:

- PEst is a 1-by-N vector of estimated parameters;
- ObjEst is the value of the objective function at optimum;

- Hess is a N-by-N approximate Hessian matrix at optimum.

If you need to use extra input arguments in your minimization function, enter a cell array instead of a plain function handle:

```
{@yourminfunc,Arg1,Arg2,...}
```

In that case, the optimiser will be called the following way:

```
[PEst,ObjEst,Hess] = yourminfunc(F,P0,PLow,PHigh,Opt,Arg1,Arg2,...)
```

User-supplied steady-state solver

You can supply a function handle to your own steady-state solver (i.e. a function that finds the steady state for given parameters) through the 'sstate=' option.

The function is expected to take one input argument, the model object with newly assigned parameters, and return at least two output arguments, the model object with a new steady state (or balanced-growth path) and a success flag. The flag is true if the steady state has been successfully computed, and false if not:

```
[M,Success] = mysstatesolver(M)
```

It is your responsibility to add the growth characteristics if some of the model variables drift over time. In other words, you need to take care of the imaginary parts of the steady state values in the model object returned by the solver.

Alternatively, you can also run the steady-state solver with extra input arguments (with the model object still being the first input argument). In that case, you need to set the option 'sstate=' to a cell array with the function handle in the first cell, and the other input arguments afterwards, e.g.

```
'sstate=',{@mysstatesolver,1,'a',X}
```

The actual function call will have the following form:

```
[M,Success] = mysstatesolver(M,1,'a',X)
```

Example

evalsystempriors

Evaluate minus log of system prior density

Syntax

```
[P,C,X] = evalsystempriors(M,S)
```

Input arguments

- `M` [model] - Model object on which current parameterisation the system priors will be evaluated.
- `S` [systempriors] - System priors objects.

Output arguments

- `P` [numeric] - Minus log of system prior density.
- `C` [numeric] - Contributions of individual prios to the overall system prior density.
- `X` [numeric] - Value of each expression defining a system property for which a prior has been defined in the system priors object, `S`.

Description

Example

---

expand

Compute forward expansion of model solution for anticipated shocks

Syntax

```
M = expand(M,K)
```

Input arguments

- M [model] - Model object whose solution will be expanded.
- K [numeric] - Number of periods ahead, t+k, up to which the solution for anticipated shocks will be expanded.

Output arguments

- M [model] - Model object with the solution expanded.

Description

Example

---

export

Save export files to disk

Backend IRIS function. No help provided.

---

fevd

Forecast error variance decomposition for model variables

Syntax

```
[X,Y,List,A,B] = fevd(M,Range,...)
[X,Y,List,A,B] = fevd(M,NPer,...)
```

Input arguments

- M [model] - Model object for which the decomposition will be computed.
- Range [numeric | char] - Decomposition date range with the first date beign the first forecast period.
- NPer [numeric] - Number of periods for which the decomposition will be computed.

Output arguments

- X [namedmat | numeric] - Array with the absolute contributions of individual shocks to total variance of each variables.
- Y [namedmat | numeric] - Array with the relative contributions of individual shocks to total variance of each variables.
- List [cellstr] - List of variables in rows of the X an Y arrays, and shocks in columns of the X and Y arrays.
- A [struct] - Database with the absolute contributions converted to time series.
- B [struct] - Database with the relative contributions converted to time series.

Options

- 'matrixFmt=' ['namedmat' | 'plain'] - Return matrices X and Y as be either namedmat P204 objects (i.e. matrices with named rows and columns) or plain numeric arrays.
- 'select=' [@all | char | cellstr] - Return FEVD for selected variables and/or shocks only; @all means all variables and shocks; this option does not apply to the output databases, A and B.

Description

Example

---

ffrf

Filter frequency response function of transition variables to measurement variables

Syntax

```
[F,List] = ffrf(M,Freq,...)
```

Input arguments

- M [model] - Model object for which the frequency response function will be computed.
- Freq [numeric] - Vector of frequencies for which the response function will be computed.

Output arguments

- F [namedmat | numeric] - Array with frequency responses of transition variables (in rows) to measurement variables (in columns).
- List [cell] - List of transition variables in rows of the F matrix, and list of measurement variables in columns of the F matrix.

Options

- 'include=' [char | cellstr | @all] - Include the effect of the listed measurement variables only; @all means all measurement variables.
- 'exclude=' [char | cellstr | empty] - Remove the effect of the listed measurement variables.
- 'maxIter=' [numeric | 500] - Maximum number of iteration when computing the steady-state Kalman filter.
- 'matrixFmt=' ['namedmat' | 'plain'] - Return matrix F as either a namedmat P204 object (i.e. matrix with named rows and columns) or a plain numeric array.
- 'select=' [@all | char | cellstr] - Return FFRF for selected variables only; @all means all variables.
- 'tolerance=' [numeric | 1e-7] - Convergence tolerance when computing the steady-state Kalman filter.

Description

Example

---

filter

Kalman smoother and estimator of out-of-likelihood parameters

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[M,Outp,V,Delta,PE,SCov] = filter(M,Inp,Range,~J,...)
```

Input arguments

- M [model] - Solved model object.
- Inp [struct | cell] - Input database from which observations for measurement variables will be taken.
- Range [numeric | char] - Date range on which the Kalman filter will be run.
- ~J [struct | empty] - Database with user-supplied time-varying paths for std deviation, corr coefficients, or medians for shocks; ~J is equivalent to using the option 'vary=', and may be omitted.

Output arguments

- M [model] - Model object with updates of std devs (if 'relative=' is true) and/or updates of out-of-likelihood parameters (if 'outoflik=' is non-empty).
- Outp [struct | cell] - Output struct with smoother or prediction data.
- V [numeric] - Estimated variance scale factor if the 'relative=' options is true; otherwise V is 1.
- Delta [struct] - Database with estimates of out-of-likelihood parameters.
- PE [struct] - Database with prediction errors for measurement variables.
- SCov [numeric] - Sample covariance matrix of smoothed shocks; the covariance matrix is computed using shock estimates in periods that are included in the option 'objrange=' and, at the same time, contain at least one observation of measurement variables.

Options

- 'ahead=' [numeric | 1] - Predictions will be computed this number of period ahead.
- 'chkFmse=' [true | false] - Check the condition number of the forecast MSE matrix in each step of the Kalman filter, and return immediately if the matrix is ill-conditioned; see also the option 'fmseCondTol='.
- 'condition=' [char | cellstr | empty] - List of conditioning measurement variables. Condition time t|t-1 prediction errors (that enter the likelihood function) on time t observations of these measurement variables.
- 'deviation=' [true | false] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [@auto | true | false] - Measurement data contain deterministic trends.

- 'output=' ['predict' | 'filter' | 'smooth'] - Return smoother data or filtered data or prediction data or any combination of them.
- 'fmseCondTol=' [eps() | numeric] - Tolerance for the FMSE condition number test; not used unless 'chkFmse=' true.
- 'initCond=' ['fixed' | 'optimal' | 'stochastic' | struct] - Method or data to initialise the Kalman filter; user-supplied initial condition must be a mean database or a mean-MSE struct.
- 'lastSmooth=' [numeric | Inf] - Last date up to which to smooth data backward from the end of the range; if Inf smoother will run on the entire range.
- 'meanOnly=' [true | false] - Return a plain database with mean data only; this option overrides the 'return*=' options, i.e. 'returnCont=', 'returnMse=', 'returnStd='.
- 'outOfLik=' [cellstr | empty] - List of parameters in deterministic trends that will be estimated by concentrating them out of the likelihood function.
- 'objFunc=' ['-loglik' | 'prederr'] - Objective function computed; can be either minus the log likelihood function or weighted sum of prediction errors.
- 'objRange=' [numeric | Inf] - The objective function will be computed on the specified range only; Inf means the entire filter range.
- 'precision=' ['double' | 'single'] - Numeric precision to which output data will be stored; all calculations themselves always run to double precision.
- 'relative=' [true | false] - Std devs of shocks assigned in the model object will be treated as relative std devs, and a common variance scale factor will be estimated.
- 'returnCont=' [true | false] - Return contributions of prediction errors in measurement variables to the estimates of all variables and shocks.
- 'returnMse=' [true | false] - Return MSE matrices for predetermined state variables; these can be used for settin up initial condition in subsequent call to another filter or jforecast.
- 'returnStd=' [true | false] - Return database with std devs of model variables.
- 'weighting=' [numeric | empty] - Weighting vector or matrix for prediction errors when 'objective=' 'prederr'; empty means prediction errors are weighted equally.

Options for models with nonlinear equations simulated in prediction step

- 'simulate=' [false | cell] - Use the backend algorithms from the simulate P137 function to run nonlinear simulation for each prediction step; specify options that will be passed into simulate when running a prediction step.

Description

The 'ahead=' and 'rollback=' options cannot be combined with one another, or with multiple data sets, or with multiple parameterisations.

Initial conditions in time domain

By default (with 'initCond=' 'stochastic'), the Kalman filter starts from the model-implied asymptotic distribution. You can change this behaviour by setting the option 'initCond=' to one of the following four different values:

- 'fixed' – the filter starts from the model-implied asymptotic mean (steady state) but with no initial uncertainty. The initial condition is treated as a vector of fixed, non-stochastic, numbers.
- 'optimal' – the filter starts from a vector of fixed numbers that is estimated optimally (likelihood maximising).
- database (i.e. struct with fields for individual model variables) – a database through which you supply the mean for all the required initial conditions, see help on model/get P100 for how to view the list of required initial conditions.
- mean-mse struct (i.e. struct with fields .mean and .mse) – a struct through which you supply the mean and MSE for all the required initial conditions.

Contributions of measurement variables to the estimates of all variables

Use the option 'returnCont=' true to request the decomposition of measurement variables, transition variables, and shocks into the contributions of each individual measurement variable. The resulting output database will include one extra subdatabase called .cont. In the .cont subdatabase, each time series will have Ny columns where Ny is the number of measurement variables in the model. The k-th column will be the contribution of the observations on the k-th measurement variable.

The contributions are additive for linearised variables, and multiplicative for log-linearised variables (log variables). The difference between the actual path for a particular variable and the sum of the contributions (or their product in the case of log varibles) is due to the effect of constant terms and deterministic trends.

Example

findeqtn

Find equations by the labels

Syntax

```
[Eqtn,Eqtn,...] = findeqtn(M,Label,Label,...)
[List,List,...] = findeqtn(M,'-rexp',Rexp,Rexp,...)
```

Input arguments

- M [model] - Model object in which the equations will be searched for.
- Label [char] - Equation label that will be searched for.
- Rexp [char] - Regular expressions that will be matched against equation labels.

Output arguments

- Eqtn [char] - First equation found with the label Label.
- List [cellstr] - List of equations whose labels match the regular expression Rexp.

Description

Example

---

findname

Find names of variables, shocks, or parameters by their descriptors

Syntax

```
[Name,Name,...] = findname(M,Desc,Desc,...)
[List,List,...] = findname(M,'-rexp',Rexp,Rexp,...)
```

Input arguments

- M [model] - Model object in which the names will be searched for.
- Desc [char] - Variable, shock, or parameter descriptors that will be searched for.
- Rexp [char] - Regular expressions that will be matched against variable, shock, and parameter descriptors.

Output arguments

- Name [char] - First name found with the descriptor Desc.
- List [cellstr] - List of names whose descriptors match the regular expression Rexp.

Description

Example

---

fisher

Approximate Fisher information matrix in frequency domain

Syntax

```
[F,FF,Delta,Freq] = fisher(M,NPer,PList,...)
```

Input arguments

- M [model] - Solved model object.
- NPer [numeric] - Length of the hypothetical range for which the Fisher information will be computed.
- PList [cellstr] - List of parameters with respect to which the likelihood function will be differentiated.

Output arguments

- `F` [numeric] - Approximation of the Fisher information matrix.
- `FF` [numeric] - Contributions of individual frequencies to the total Fisher information matrix.
- `Delta` [numeric] - Kronecker delta by which the contributions in `Fi` need to be multiplied to sum up to `F`.
- `Freq` [numeric] - Vector of frequencies at which the Fisher information matrix is evaluated.

Options

- `'chkSstate='` [`true` | `false` | cell] - Check steady state in each iteration; works only in non-linear models.
- `'deviation='` [`true` | `false`] - Exclude the steady state effect at zero frequency.
- `'exclude='` [char | cellstr | `empty`] - List of measurement variables that will be excluded from the likelihood function.
- `'percent='` [`true` | `false`] - Report the overall Fisher matrix `F` as Hessian w.r.t. the log of variables; the interpretation for this is that the Fisher matrix describes the changes in the log-likelihood function in reponse to percent, not absolute, changes in parameters.
- `'progress='` [`true` | `false`] - Display progress bar in the command window.
- `'solve='` [`true` | `false` | cellstr] - Re-compute solution in each differentiation step; you can specify a cell array with options for the `solve` function.
- `'sstate='` [`true` | `false` | cell] - Re-compute steady state in each differentiation step; if the model is non-linear, you can pass in a cell array with opt used in the `sstate` function.

Description

Example

---

fmse

Forecast mean square error matrices

Syntax

```
[F,List,D] = fmse(M,NPer,...)
[F,List,D] = fmse(M,Range,...)
```

Input arguments

- `M` [model] - Model object for which the forecast MSE matrices will be computed.
- `NPer` [numeric] - Number of periods.
- `Range` [numeric | char] - Date range.

Output arguments

- `F` [namedmat | numeric] - Forecast MSE matrices.
- `List` [cellstr] - List of variables in rows and columns of `M`.
- `D` [dbase] - Database with the std deviations of individual variables, i.e. the square roots of the diagonal elements of `F`.

Options

- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrix `F` as either a `namedmat` P204 object (i.e. matrix with named rows and columns) or a plain numeric array.
- `'select='` [`@all` | char | cellstr] - Return FMSE for selected variables only; `@all` means all variables.

Description

Example

■ get

Query model object properties

Syntax

```
Ans = get(M,Query)
[Ans,Ans,...] = get(M,Query,Query,...)
```

Input arguments

- `M` [model] - Model object.
- `Query` [char] - Query to the model object.

Output arguments

- `Ans` [...] - Answer to the query.

Valid queries to model objects

This is the categorised list of queries to model objects. Note that letter `'y'` is used in various contexts to denote measurement variables or equations, `'x'` transition variables or equations, `'e'` shocks, `'p'` parameters, `'g'` exogenous variables, `'d'` deterministic trend equations, and `'l'` dynamic links. The property names are case insensitive.

Steady state

- `'sstate'` – Returns [struct] a database with the steady states for all model variables. The steady states are described by complex numbers in which the real part is the level and the imaginary part is the growth rate.
- `'sstateLevel'` – Returns [struct] a database with the steady-state levels for all model variables.
- `'sstateGrowth'` – Returns [struct] a database with steady-state growth (first difference for linearised variables, gross rate of growth for log-linearised variables) for all model variables.
- `'dtrends'` – Returns [struct] a database with the effect of the deterministic trends on the measurement variables. The effect is described by complex numbers the same way as the steady state.
- `'dtrendsLevel'` – Returns [struct] a database with the effect of the deterministic trends on the steady-state levels of the measurement variables.
- `'dtrendsGrowth'` – Returns [struct] a database with the effect of deterministic trends on steady-state growth of the measurement variables.

- 'sstate+dtrends' – Returns [struct] the same as 'sstate' except that the measurement variables are corrected for the effect of the deterministic trends.
- 'sstateLevel+dtrendsLevel' – Returns [struct] the same as 'sstateLevel' except that the measurement variables are corrected for the effect of the deterministic trends.
- 'sstateGrowth+dtrendsGrowth' – Returns [struct] the same as 'sstateGrowth' except that the measurement variables are corrected for the effect of the deterministic trends.

Variables, shocks, and parameters

- 'yList', 'xList', 'eList', 'pList', 'gList' - Return [cellstr] the lists of, respectively, measurement variables (y), transition variables (x), shocks (e), parameters (p), and exogenous variables (g), each in order of appearance of the names in declaration sections of the original model file. Note that the list of parameters, 'pList', does not include the names of std deviations or cross-correlations.
- 'eyList' – Returns [cellstr] the list of measurement shocks in order of their appearance in the model code declarations; only those shocks that actually occur in at least one measurement equation are returned.
- 'exList' – Returns [cellstr] the list of transition shocks in order of their appearance in the model code declarations; only those shocks that actually occur in at least one transition equation are returned.
- 'stdList' – Returns [cellstr] the list of the names of the standard deviations for the shocks in order of the appearance of the corresponding shocks in the model code.
- 'corrList' – Returns [cellstr] the list of the names of cross-correlation coefficients for the shocks in order of the appearance of the corresponding shocks in the model code.
- 'stdCorrList' – Returns [cellstr] the list of the names of std deviations and cross-correlation coefficients for the shocks in order of the appearance of the corresponding shocks in the model code.

Equations

- 'yEqtn', 'xEqtn', 'dEqtn', 'lEqtn' - Return [cellstr] the lists of, respectively, to measurement equations (y), transition equations (x), deterministic trends (d), and dynamic links (l), each in order of appearance in the original model file.
- 'links' – Returns [struct] a database with the dynamic links with fields names after the LHS name.
- 'rpteq' – Returns [rpteq] a reporting equations (rpteq) object (if !reporting_equations were included in the model file).

First-order Taylor expansion of equations

- 'derivatives' – Returns [cellstr] the symbolic/automatic derivatives for each model equation; in each equation, the derivatives w.r.t. all variables present in that equation are evaluated at once and returned as a vector of numbers; see also 'wrt'.
- 'wrt' - Returns [cellstr] the list of the variables (and their auxiliary lags or leads) with respect to which the corresponding equation in 'derivatives' is differentiated.

Descriptions and aliases of variables, parameters, and shocks

- 'descript' – Returns [struct] a database with user descriptions of model variables, shocks, and parameters.
- 'yDescript', 'xDescript', 'eDescript', 'pDescript', 'gDescript' - Return [cellstr] user descriptions of, respectively, measurement variables (y), transition variables (x), shocks (e), parameters (p), and exogenous variables (g).
- 'alias' – Returns [struct] a database with all aliases of model variables, shocks, and parameters.
- 'yAlias', 'xAlias', 'eAlias', 'pAlias', 'gAlias' - Return [cellstr] the aliases of, respectively, measurement variables (y), transition variables (x), shocks (e), parameters (p), and exogenous variables (g).

Equation labels and aliases

- 'labels' – Returns [cellstr] the list of all user labels added to equations.
- 'yLabels', 'xLabels', 'dLabels', 'lLabels', 'rLabels' - Return [cellstr] user labels added, respectively, to measurement equations (y), transition equations (x), deterministic trends (d), and dynamic links (l).
- 'eqtnAlias' – Returns [cellstr] the list of all aliases added to equations.
- 'yEqtnAlias', 'xEqtnAlias', 'dEqtnAlias', 'lEqtnAlias', 'rEqtnAlias' - Return [cellstr] the aliases of, respectively, measurement equations (y), transition equations (x), deterministic trends (d), and dynamic links (l).

Parameter values

- 'corr' – Returns [struct] a database with current cross-correlation coefficients of shocks.
- 'nonzeroCorr' – Returns [struct] a database with current nonzero cross-correlation coefficients of shocks.
- 'parameters' – Returns [struct] a database with current parameter values, including the std devs and non-zero corr coefficients.
- 'std' – Returns [struct] a database with current std deviations of shocks.

Eigenvalues

- 'stableRoots' – Returns [cell of numeric] a vector of the model eigenvalues that are smaller than one in magnitude (allowing for rounding errors around one).
- 'unitRoots' – Returns [cell of numeric] a vector of the model eigenvalues that equal one in magnitude (allowing for rounding errors around one).
- 'unstableRoots' [cell of numeric] A vector of the model eigenvalues that are greater than one in magnitude (allowing for rounding errors around one).

Model structure, solution, build

- 'build' – Returns [numeric] IRIS version number under which the model object has been built.
- 'eqtnBlk' – Returns [cell] of cell str with the recursive block structure of steady-state equations (if the block-recursive analysis has already been performed).
- 'log' – Returns [struct] a database with true for each log-linearised variables, and false for each linearised variable.
- 'maxLag' – Returns [numeric] the maximum lag in the model.
- 'maxLead' – Returns [numeric] the maximum lead in the model.
- 'nameBlk' – Returns [cell] of cell str with the recursive block structure of variable names (if the block-recursive analysis has already been performed).
- 'stationary' – Returns [struct] a database with true for each stationary variables, and false for each unit-root (non-stationary) variables (under current solution).
- 'nonStationary' – Returns [struct] a database with true for each unit-root (non-stationary) varible, and false for each stationary variable (under current solution).
- 'stationaryList' – Returns [cellstr] the list of stationary variables (under current solution).
- 'nonStationaryList' – Returns [cellstr] cell with the list of unit-root (non-stationary) variables (under current solution).
- 'initCond' – Returns [cellstr] the list of the lagged transition variables that need to be supplied as initial conditions in simulations and forecasts. The list of the initial conditions is solution-specific as the state-spece coefficients at some of the lags may evaluate to zero depending on the current parameters.
- 'yVector' – Returns [cellstr] the list of measurement variables in order of their appearance in the rows and columns of state-space matrices (effectively identical to 'yList') from the model/sspace P145 function.

- 'xVector' – Returns [cellstr] the list of transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the model/sspace P145 function.
- 'xfVector' – Returns [cellstr] the list of forward-looking (i.e. non-predetermined) transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the model/sspace P145 function.
- 'xbVector' – Returns [cellstr] the list of backward-looking (i.e. predetermined) transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the model/sspace P145 function.
- 'eVector' – Returns [cellstr] the list of the shocks in order of their appearance in the rows and columns of state-space matrices (effectively identical to 'eList') from the model/sspace P145 function.

Description

First-order Taylor expansion of equations

The expressions for symbolic/automatic derivatives of individual model equations returned by 'derivatives' are expressions that evaluate the derivatives with respect to all variables present in that equation at once. The list of variables with respect to which each equation is differentiated is returned by 'wrt'.

The expressions returned by the query 'derivatives' can refer to

- the names of model parameters, such as alpha;
- the names of transition or measurement variables, such as X;
- the lags or leads of variables, such as X{-1} or X{2}.

Note that the lags and leads of variables must be, in general, preserved in the derivatives for non-stationary (unit-root) models. For stationary models, the lags and leads can be removed and each simply replaced with the current date of the respective variable.

Example

```
d = get(m,'derivatives');
w = get(m,'wrt');
```

The 1-by-N cell array d (where N is the total number of equations in the model) will contain expressions that evaluate to the vector of derivatives of the individual equations w.r.t. to the variables present in that equation:

```
d{k}
```

is an expression that returns, in general, a vector of M numbers. These M numbers are the derivatives of the k-th equation w.r.t to M variables whose list is in

```
w{k}
```

---

horzcat

Combine two compatible model objects in one object with multiple parameterisations

Syntax

```
M = [M1,M2,...]
```

Input arguments

- `M1`, `M2` [model] - Compatible model objects that will be combined; the input models must be based on the same model file.

Output arguments

- `M` [model] - Output model object that combines the input model objects as multiple parameterisations.

Description

Example

---

icrf

Initial-condition response functions

Syntax

```
S = icrf(M,NPer,...)
S = icrf(M,Range,...)
```

Input arguments

- `M` [model] - Model object for which the initial condition responses will be simulated.
- `Range` [numeric | char] - Date range with the first date being the shock date.
- `NPer` [numeric] - Number of periods.

Output arguments

- `S` [struct] - Database with initial condition response series.

Options

- `'delog='` [`true` | `false`] - Delogarithmise the responses for variables declared as `!log_variables`.
- `'size='` [numeric | `1` for linear models | `log(1.01)` for non-linear models] - Size of the deviation in initial conditions.

Description

Example

■ ifrf

Frequency response function to shocks

Syntax

```
[W,List] = ifrf(M,Freq,...)
```

Input arguments

- M [model] - Model object for which the frequency response function will be computed.
- Freq [numeric] - Vector of frequencies for which the response function will be computed.

Output arguments

- W [namedmat | numeric] - Array with frequency responses of transition variables (in rows) to shocks (in columns).
- List [cell] - List of transition variables in rows of the W matrix, and list of shocks in columns of the W matrix.

Options

- 'matrixFmt=' ['namedmat' | 'plain'] - Return matrix W as either a namedmat P204 object (i.e. matrix with named rows and columns) or a plain numeric array.
- 'select=' [@all | char | cellstr] - Return IFRF for selected variables only; @all means all variables.

Description

Example

■ iscompatible

True if two models can occur together on the LHS and RHS in an assignment

Syntax

```
Flag = iscompatible(M1,M2)
```

Input arguments

- M1, M2 [model] - Two model objects that will be tested for compatibility.

Output arguments

- Flag [true | false] - True if M1 and M1 can occur in an assignment, M1(...) = M2(...) or horziontal concatenation, [M1,M2].

Description

The function compares the names of all variables, shocks, and parameters, and the composition of the state-space vectors.

Example

---

islinear

True for models declared as linear

Syntax

```
Flag = islinear(M)
```

Input arguments

- m [model] - Queried model object.

Output arguments

- Flag [true | false] - True if the model has been declared linear.

Description

The value returned dependes on whether the model has been declared as linear by the user when constructing the model object by calling the model/model P122 function. In other words, no check is performed whether or not the model is actually linear.

Example

```
m = model('mymodel.file','linear=',true);
islinear(m)
ans =
     1
```

---

islocked

Get lock status of dynamic links or sstate update equations

Syntax

```
List = islocked(M,'!links')
Flag = islocked(M,'!links',Name);
List = islocked(M,'!sstate_update');
Flag = islocked(M,'!sstate_update',Name);
```

Input arguments

- `M` [model] - Model object.
- `Name` [char] - Name of LHS variable in links or sstate update equations whose status will be returned.

Output arguments

- `List` [cellstr] - List of LHS names in `!links` P36 or `!sstate_update` P44 equations that are currently locked in the model object `M`.
- `Flag` [`true` | `false`] - Lock status (`true` if locked, `false` if not locked) of LHS name `Name` in `!links` P36 or `!sstate_update` P44 equations.

Example

---

islog

True for log-linearised variables

Syntax

```
Flag = islog(M,Name)
```

Input arguments

- M [model] - Model object.
- Name [char | cellstr] - Name or names of model variable(s).

Output arguments

- Flag [true | false] - True for log variables.

Description

Example

---

ismissing

True if some initical conditions are missing from input database

Syntax

```
[Flag,List] = ismissing(M,Inp,Range)
```

Input arguments

- M [model] - Model object.
- Inp [struct] - Input database from which initical conditions are obtained.
- Range [numeric] - Simulation range.%

Output arguments

- Flag [true | false] - True if one or more initial conditions required for simulation of the model M are missing from the database Inp.
- List [cellstr] - List of initial conditions missing from the database Inp.

Description

The complete list of initial conditions required for simulating the model M can be obtained by

```
get(M,'required')
```

Example

---

isname

True for valid names of variables, parameters, or shocks in model object

Syntax

```
Flag = isname(M,Name)
[Flag,Flag,...] = isname(M,Name,Name,...)
```

Input arguments

- M [model] - Model object.
- Name [char] - A text string that will be matched against the names of variables, parameters and shocks in the model object M.

Output arguments

- Flag [true | false] - True for input strings that are valid names in the model object M.

Description

Example

---

isnan

Check for NaNs in model object

Syntax

```
[Flag,List] = isnan(M,'parameters')
[Flag,List] = isnan(M,'sstate')
[Flag,List] = isnan(M,'derivatives')
[Flag,List] = isnan(M,'solution')
```

Input arguments

- `M` [model] - Model object.

Output arguments

- `Flag` [`true` | `false`] - True if at least one `NaN` value exists in the queried category.
- `List` [cellstr] - List of parameters (if called with `'parameters'`) or variables (if called with `'sstate'`) that are assigned NaN in at least one parameterisation, or equations (if called with `'derivatives'`) that produce an NaN derivative in at least one parameterisation.

Description

Example

---

issolved

True if model solution exists

Syntax

```
Flag = issolved(M)
```

Input arguments

- `M` [model] - Model object.

Output arguments

- `Flag` [`true` | `false`] - True for each parameterisation for which a stable unique solution exists currently in the model object.

Description

Example

---

isstationary

True if model or specified combination of variables is stationary

Syntax

```
Flag = isstationary(M)
Flag = isstationary(M,Name)
Flag = isstationary(M,LinComb)
```

Input arguments

- `M` [model] - Model object.
- `Name` [char] - Transition variable name.
- `LinComb` [char] - Text string defining a linear combination of transition variables; log variables need to be enclosed in `log(...)`.

Output arguments

- Flag [true | false] - True if the model (if called without a second input argument) or the specified transition variable or combination of transition variables (if called with a second input argument) is stationary.

Description

Example

In the following examples, m is a solved model object with two of its transition variables named X and Y; the latter is a log variable:

```
isstationary(m)
isstationary(m,'X')
isstationary(m,'log(Y)')
isstationary(m,'X - 0.5*log(Y)')
```

■ jforecast

Forecast with judgmental adjustments (conditional forecasts)

Syntax

```
F = jforecast(M,D,Range,...)
```

Input arguments

- M [model] - Solved model object.
- D [struct] - Input data from which the initial condition is taken.
- Range [numeric] - Forecast range.

Output arguments

- F [struct] - Output struct with the judgmentally adjusted forecast.

Options

- 'anticipate=' [true | false] - If true, real future shocks are anticipated, imaginary are unanticipated; vice versa if false.
- 'currentOnly=' [true | false] - If true, MSE matrices will be computed only for the current-dated variables, not for their lags or leads (expectations).
- 'deviation=' [true | false] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [@auto | true | false] - Measurement data contain deterministic trends.
- 'initCond=' ['data' | 'fixed'] - Use the MSE for the initial conditions if found in the input data or treat the initical conditions as fixed.
- 'meanOnly=' [true | false] - Return only mean data, i.e. point estimates.
- 'plan=' [plan] - Simulation plan specifying the exogenised variables and endogenised shocks.
- 'vary=' [struct | empty] - Database with time-varying std deviations or cross-correlations of shocks.

Description

When adjusting the mean and/or std devs of shocks, you can use real and imaginary numbers ot distinguish between anticipated and unanticipated shocks:

- any shock entered as an imaginary number is treated as an anticipated change in the mean of the shock distribution;
- any std dev of a shock entered as an imaginary number indicates that the shock will be treated as anticipated when conditioning the forecast on the reduced-form tunes.
- the same shock or its std dev can have both the real and the imaginary part.

Description

Example

---

length

Number of alternative parameterisations

Syntax

```
N = length(M)
```

Input arguments

- M [model | esteq] - Model or esteq object.

Output arguments

- N [numeric] - Number of alternative parameterisations.

Description

Example

---

lhsmrhs

Evaluate the discrepancy between the LHS and RHS for each model equation and given data

Syntax for casual evaluation

```
Q = lhsmrhs(M,D,Range)
```

Syntax for fast evaluation

```
Q = lhsmrhs(M,YXE)
```

Input arguments

M [model] - Model object whose equations and currently assigned parameters will be evaluated.

YXE [numeric] - Numeric array created from an input database by calling the function data4lhsmrhs P81; YXE contains the observations on the measurement variables, transition variables, and shocks organised row-wise.

- D [struct] - Input database with observations on measurement variables, transition variables, and shocks on which the discrepancies will be evaluated.
- Range [numeric] - Date range on which the discrepancies will be evaluated.

Output arguments

Q [numeric] - Numeric array with discrepancies between the LHS and RHS for each model equation.

Description

The function lhsmrhs evaluates the discrepancy between the LHS and the RHS in each model equation; each lead is replaced with the actual observation supplied in the input data. The function lhsmrhs does not work for models with references to steady state values P53.

The first syntax, with the array YXE pre-built in a call to data4lhsmrhs P81 is computationally much more efficient if you need to evaluate the LHS-RHS discrepancies repeatedly for different parameterisations.

The output argument D is an nEqtn by nPer by nAlt array, where nEqnt is the number of measurement and transition equations, nPer is the number of periods used to create X in a prior call to data4lhsmrhs P81, and nAlt is the greater of the number of alternative parameterisations in M, and the number of alternative datasets in the input data.

Example

```
YXE = data4lhsmrhs(M,d,range);
Q = lhsmrhs(M,YXE);
```

---

lock

Lock (disable) dynamic links or sstate update equations temporarily

Syntax

```
M = lock(M,'!links')
M = lock(M,'!links',Name1,Name2,...);
M = lock(M,'!sstate_update');
M = lock(M,'!sstate_update',Name1,Name2,...);
```

Input arguments

- M [model] - Model object.
- Name1, Name2 [char] - List of names whose links or sstate update equations will be temporarily locked.

Output arguments

- M [model] - Model object with dynamic links !links P36 or steady-state update equations !sstate_update P44 temporarily disabled, until unlock P156.

Example

---

loglik

Evaluate minus the log-likelihood function in time or frequency domain

Full syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[Obj,V,F,PE,Delta,PDelta] = loglik(M,Inp,Range,~J,...)
```

Syntax for fast one-off likelihood evaluation

Input arguments marked with a ~ (tilde) sign may be omitted.

```
Obj = loglik(M,Inp,Range,~J,...)
```

Syntax for repeated fast likelihood evaluations

Input arguments marked with a ~ (tilde) sign may be omitted.

```
% Step #1: Initialise.
loglik(M,Inp,Range,~J,...,'persist=',true);
```

```
% Step #2: Assign/change parameters.
M... = ...; % Change parameters.

% Step #3: Re-compute steady state and solution if necessary.
M = ...;
M = ...;

% Step #4: Evaluate likelihood.
L = loglik(M);

% Repeat steps #2, #3, #4 for different values of parameters.
% ...
```

Input arguments

- `M` [model] - Solved model object.
- `Inp` [struct | cell] - Input database from which observations for measurement variables will be taken.
- `Range` [numeric | char] - Date range on which the Kalman filter will be run.
- `~J` [struct | empty] - Database with user-supplied time-varying paths for std deviation, corr coefficients, or medians for shocks; `~J` is equivalent to using the option `'vary='`, and may be omitted.

Output arguments

- `Obj` [numeric] - Value of minus the log-likelihood function (or other objective function if specified in options).
- `V` [numeric] - Estimated variance scale factor if the `'relative='` options is true; otherwise `V` is 1.
- `F` [numeric] - Sequence of forecast MSE matrices for measurement variables.
- `PE` [struct] - Database with prediction errors for measurement variables; exp of prediction errors for measurement variables declared as log variables.
- `Delta` [struct] - Databse with point estimates of the deterministic trend parameters specified in the `'outoflik='` option.
- `PDelta` [numeric] - MSE matrix of the estimates of the `'outoflik='` parameters.

Options

- 'objDecomp=' [true | false] - Decompose the objective function into the contributions of individual time periods (in time domain) or individual frequencies (in frequency domain); the contributions are added as extra rows in the output argument Obj.
- 'persist=' [true | false] – Pre-process and store the overhead (data and options) for subsequent fast calls.

See help on model/filter P93 for other options available.

Description

The number of output arguments you request when calling loglik affects computational efficiency. Running the function with only the first output argument, i.e. the value of the likelihood function (minus the log of it, in fact), results in the fastest performance.

The loglik function runs an identical Kalman filter as model/filter P93, the only difference is the types and order of output arguments returned.

Fast evaluation of likelihood

Every time you change the parameters, you need to update the steady state and solution of the model if necessary by yourself, before calling loglik. Follow these rules:

- If you only change std deviations and no other parameters, you don't have to re-calculate steady state or solution.
- If the model is linear, you only need to call solve P143.
- The only exception to rules #2 and #3 is when the model has dynamic links P36 with references to some steady state values. In that case, you must also run sstate P147 after solve P143 in linear models to update the steady state.
- If the model is non-linear, and you only change parameters that affect transitory dynamics and not the steady state, you only need to call solve P143.
- If the model is non-linear, and you change parameters that affect both transitory dynamics and steady state, you must run first sstate P147 and then solve P143.

Example

lognormal

Characteristics of log-normal distributions returned from filter of forecast

Syntax

```
D = lognormal(M,D,...)
```

Input arguments

- M [model] - Model on which the filter or forecast function has been run.
- D [struct] - Struct or database returned from the filter or forecast function.

Output arguments

- D [struct] - Struct including new sub-databases with requested log-normal statistics.

Options

- 'fresh=' [true | false] - Output structure will include only the newly computed databases.
- 'mean=' [true | false] - Compute the mean of the log-normal distributions.
- 'median=' [true | false] - Compute the median of the log-normal distributions.
- 'mode=' [true | false] - Compute the mode of the log-normal distributions.
- 'prctile=' [numeric | [5,95]] - Compute the selected percentiles of the log-normal distributions.
- 'prefix=' [char | 'lognormal'] - Prefix used in the names of the newly created databases.
- 'std=' [true | false] - Compute the std deviations of the log-normal distributions.

Description

---

model

Create new model object based on model file

Syntax

```
M = model(FName,...)
M = model(M,...)
```

Input arguments

- FName [char | cellstr] - Name(s) of model file(s) that will be loaded and converted to a new model object.
- M [model] - Existing model object that will be rebuilt as if from a model file.

Output arguments

- M [model] - New model object based on the input model code file or files.

Options

- 'assign=' [struct | empty] - Assign model parameters and/or steady states from this database at the time the model objects is being created.
- 'baseYear=' [numeric | 2000] - Base year for constructing deterministic time trends.
- 'blazer=' [true | false] - Perform block-recursive analysis of steady-state equations at the time the model object is being created; the option works only in nonlinear models.
- 'comment=' [char | empty] - Text comment attached to the model object.
- 'declareParameters=' [true | false] - If false, skip parameter declaration in the model file, and determine the list of parameters automatically as names found in equations but not declared.
- 'epsilon=' [numeric | eps^(1/4)] - The minimum relative step size for numerical differentiation.
- 'linear=' [true | false] - Indicate linear models.
- 'makeBkw=' [@auto | @all | cellstr | char] - Variables included in the list will be made part of the vector of backward-looking variables; @auto means the variables that do not have any lag in model equations will be put in the vector of forward-looking variables.
- 'multiple=' [true | false] - Allow each variable, shock, or parameter name to be declared (and assigned) more than once in the model file.

- 'optimal=' [cellstr] - Specify optimal policy options, see below; only applies when the keyword min P58 is used in the model file.
- 'removeLeads=' [true | false] - Remove all leads from the state-space vector, keep included only current dates and lags.
- 'sstateOnly=' [true | false] - Read in only the steady-state versions of equations (if available).
- 'std=' [numeric | @auto] - Default standard deviation for model shocks; @auto means 1 for linear models and log(1.01) for nonlinear models.
- 'userdata=' [... | empty] - Attach user data to the model object.

Optimal policy options

- 'multiplierPrefix=' [char | 'Mu_'] - Prefix used to create names for lagrange multipliers associated with the optimal policy problem; the prefix is followed by the equation number.
- 'nonnegative=' [cellstr] - List of variables constrained to be nonnegative.
- 'type=' ['commitment' | 'discretion'] - Type of optimal policy.

Description

Loading a model file ————————————-

The model function can be used to read in a model file P21 named fname, and create a model object m based on the model file. You can then work with the model object in your own m-files, using using the IRIS model functions P65 and standard Matlab functions.

If fname is a cell array of more than one file names then all files are combined together in order of appearance.

Re-building an existing model object ——————————————

The only instance where you may need to call a model function on an existing model object is to change the 'removeLeads=' option. Of course, you can always achieve the same by loading the original model file.

Example

Read in a model code file named my.model, and declare the model as linear:

```
m = model('my.model','linear',true);
```

Example

Read in a model code file named my.model, declare the model as linear, and assign some of the model parameters:

```
m = model('my.model','linear=',true,'assign=',P);
```

Note that this is equivalent to

```
m = model('my.model','linear=',true);
m = assign(m,P);
```

unless some of the parameters passed in to the model fuction are needed to evaluate if P33 or !switch P45 expressions.

---

neighbourhood

Evaluate the local behaviour of the objective function around the estimated parameter values

Syntax

```
[D,FigH,AxH,ObjH,LikH,EstH,BH] = neighbourhood(M,PS,Neigh,...)
```

Input arguments

- M [model | bkwmodel] - Model or bkwmodel object.
- PS [poster] - Posterior simulator (poster) object returned by the model/estimate P85 function.
- Neigh [numeric] - The neighbourhood in which the objective function will be evaluated defined as multiples of the parameter estimates.

Output arguments

- D [struct] - Struct describing the local behaviour of the objective function and the data likelihood (minus log likelihood) within the specified range around the optimum for each parameter.

The following output arguments are non-empty only if you choose 'plot=' true:

- FigH [numeric] - Handles to the figures created.
- AxH [numeric] - Handles to the axes objects created.
- ObjH [numeric] - Handles to the objective function curves plotted.
- LikH [numeric] - Handles to the data likelihood curves plotted.
- EstH [numeric] - Handles to the actual estimate marks plotted.
- BH [numeric] - Handles to the bounds plotted.

Options

- 'plot=' [true | false] - Call the grfun.plotneigh P480 function from within neighbourhood to visualise the results.
- 'neighbourhood=' [struct | empty] - Struct specifying the neighbourhood points for some of the parameters; these points will be used instead of those based on Neigh.

Plotting options

See help on grfun.plotneigh P480 for options available when you choose 'plot=' true.

Description

In the output database, D, each parameter is a 1-by-3 cell array. The first cell is a vector of parameter values at which the local behaviour was investigated. The second cell is a matrix with two column vectors: the values of the overall minimised objective function (as set up in the estimate P85 function), and the values of the data likelihood component. The third cell is a vector of four numbers: the parameter estimate, the value of the objective function at optimum, the lower bound and the upper bound.

Example

■ omega

Get or set the covariance matrix of shocks

Syntax for getting covariance matrix

```
OMG = omega(M)
```

Syntax for setting covariance matrix

```
M = omega(M,OMG)
```

Input arguments

- M [model | bkwmodel] - Model or bkwmodel object.
- OMG [numeric] - Covariance matrix that will be converted to new values for std deviations and cross-corr coefficients.

Output arguments

- OMG [numeric] - Covariance matrix of shocks or residuals based on the currently assigned std deviations and cross-correlation coefficients.
- M [model | bkwmodel] - Model or bkwmodel object with new values for std deviations and cross-corr coefficients based on the input covariance matrix.

Description

Example

---

refresh

Refresh dynamic links

Syntax

```
M = refresh(M)
```

Input arguments

- M [model] - Model object whose dynamic links will be refreshed.

Output arguments

- M [model] - Model object with dynamic links refreshed.

Description

Example

```
m = refresh(m);
```

---

regress

Centred population regression for selected model variables

Syntax

```
[B,CovRes,R2] = regress(M,Lhs,Rhs,...)
```

Input arguments

- M [model] - Model on whose covariance matrices the popolation regression will be based.
- Lhs [char | cellstr] - Lhs variables in the regression; each of the variables must be part of the state-space vector.
- Rhs [char | cellstr] - Rhs variables in the regression; each of the variables must be part of the state-space vector, or must refer to a larger lag of a transition variable present in the state-space vector.

Output arguments

- B [namedmat | numeric] - Population regression coefficients.
- CovRes [namedmat | numeric] - Covariance matrix of residuals from the population regression.
- R2 [numeric] - Coefficient of determination (R-squared).

Options

- 'matrixFmt=' ['namedmat' | 'plain'] - Return matrices B and CovRes as either namedmat P204 object (i.e. matrices with named rows and columns) or plain numeric arrays.

Description

Population regressions calculated by this function are always centred. This means the regressions are always calculated as if estimated on observations with their uncondional means (the steady-state levels) removed from them.

The Lhs and Rhs variables that are log variables must include log( ) explicitly in their names. For instance, if X is declared to be a log variable, then you must refer to log(X) or log(X{-1}).

Example

```
[B,C] = regress('log(R)',{'log(R{-1})','log(dP)'});
```

---

reporting

Evaluate reporting equations from within model object

Syntax

```
D = reporting(M,D,Range,...)
```

Input arguments

- M [model] - Model object with reporting equations.
- D [struct] - Input database that will be used to evaluate the reporting equations.
- Range [numeric | char] - Date range on which the reporting equations will be evaluated.

Output arguments

- D [struct] - Output database with reporting variables.

Options

See rpteq/run P164 for options available.

Description

■ resample

Resample from the model implied distribution

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
Oupt = resample(M,~Inp,Range,~NDraw,~J,...)
```

Input arguments

- M [model] - Solved model object with single parameterization.
- Inp [struct | empty] - Input data (if needed) for the distributions of initial condition and/or empirical shocks.
- Range [numeric | char] - Resampling date range.
- ~NDraw [numeric | 1] - Number of draws; may be omitted.
- ~J [struct | [ ]] - Database with user-supplied time-varying paths for std deviation, corr coefficients, or medians for shocks; ~J is equivalent to using the option 'vary=', and may be omitted.

Output arguments

- Outp [struct] - Output database with resampled data.

Options

- 'bootstrapMethod=' ['efron' | 'wild' | numeric] - Numeric options correspond to block sampling methods. Use a positive integer to specify a fixed block length, or a value strictly between zero and one to specify random block lengths based on a geometric distribution.
- 'deviation=' [true | false] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [@auto | true | false] - Add deterministic trends to measurement variables.
- 'method=' ['bootstrap' | 'montecarlo'] - Method of randomising shocks and initial condition.
- 'progress=' [true | false] - Display progress bar in the command window.
- 'randomInitCond=' [true | false | numeric] - Randomise initial condition; a number means the initial condition will be simulated using the specified number of extra pre-sample periods.
- 'stateVector=' ['alpha' | 'x'] - When resampling initial condition, use the transformed state vector, alpha, or the vector of original variables, x; this option is meant to guarantee replicability of results.
- 'svdOnly=' [true | false] - Do not attempt Cholesky and only use SVD to factorize the covariance matrix when resampling initial condition; only applies when 'randomInitCond=' true.

Description

When you use wild bootstrap for resampling the initial condition, the results are based on an assumption that the mean of the initial condition is the asymptotic mean implied by the model (i.e. the steady state).

References

1. Politis, D. N., & Romano, J. P. (1994). The stationary bootstrap. Journal of the American Statistical Association, 89(428), 1303-1313.

Example

■ reset

Reset specific values within model object

Syntax

```
M = reset(M)
M = reset(M,Req1,Req2,...)
```

Input arguments

- `M` [model] - Model object in which the requested type(s) of values will be reset.
- `Req1`, `Req2`, ... [`'corr'` | `'parameters'` | `'sstate'` | `'std'` | `'stdcorr'`] - Requested type(s) of values that will be reset; if omitted, everything will be reset.

Output arguments

- `M` [model] - Model object with the requested values reset.

Description

- `'corr'` - All cross-correlation coefficients will be reset to `0`.
- `'parameters'` - All parameters will be reset to `NaN`.
- `'sstate'` - All steady state values will be reset to `NaN`.
- `'std'` - All std deviations will be reset to `1` (in linear models) or `log(1.01)` (in non-linear models).
- `'stdcorr'` - Equivalent to `'std'` and `'corr'`.

Example

-IRIS Toolbox. -Copyright (c) 2007-2015 IRIS Solutions Team.

■ rollback

Prepare database for a rollback run of Kalman filter

Syntax

```
Inp = rollback(M,Inp,Range,Date)
```

Input argument

- M [model] - Model object with a single parameterization.
- Inp [struct] - Database with a single set of input data for a Kalman filter.
- Range [numeric | char] - Filter data range.
- Date [numeric] - Date up to which the input data entries will be rolled back, see Description.

Output argument

- Inp [struct] - New database with new data sets added to each tseries for measurement variables, taking out one observation at a time, see Description.

Description

The function rollback takes a database with a single set of input data that is supposed to be used in a future call to a Kalman filter, model/filter P93, and creates additional data sets (i.e. addition columns in tseries for measurement variables contained in the database) in the following way:

- the total number of the new data sets (new columns added to each measurement tseries) is N = NPer*Ny where NPer is the number of rollback periods, from Date to the end of Range (including both), and Ny is the number of measurement variables in the model M.
- The first additional data set is created by removing the observation on the last measurement variable in the last period (i.e. end of Range) and replacing it with a NaN.
- The second additional data set is created by removing the observatoins on the last two measurement variables in the last period, and so on.
- The N-th (last) additional data set is created by removing all observations in all periods between Data and end of Range.

Example

If the model m contains, for instance, 3 measurement variable, the following commands will produce a total of 13 Kalman filter runs, the first one on the original database d, and the other 12 on the rollback data sets, with individual observations removed one by one:

```
dd = rollback(m,d,qq(2000,1):qq(2015,4),qq(2015,1));
[mf,f] = filter(m,dd,qq(2000,1):qq(2015,4));
```

set

Change modifiable model object property

Syntax

```
M = set(M,Request,Value)
M = set(M,Request,Value,Request,Value,...)
```

Input arguments

- `M` [model] - Model object.
- `Request` [char] - Name of a modifiable model object property that will be changed.
- `Value` [...] - Value to which the property will be set.

Output arguments

- `M` [model] - Model object with the requested property or properties modified.

Valid requests to model objects

Equation labels and aliases

- `'yLabels='`, `'xLabels='`, `'dLabels='`, `'lLabels='` [cellstr] - Change the labels attached to, respectively, measurement equations (`y`), transition equations (`x`), deterministic trends (`d`), and dynamic links (`d`).
- `'labels='` [cell] - Change the labels attached to all equations; needs to be a cellstr matching the size of `get(M,'labels')`.
- `'yeqtnAlias='`, `'xeqtnAlias='`, `'deqtnAlias='`, `'leqtnAlias='` [cellstr] - Change the aliases of, respectively, measurement equations (`y`), transition equations (`x`), deterministic trends (`d`), and dynamic links (`d`).
- `'eqtnAlias='` [cell] - Change the aliases of all equations; needs to be a cellstr matching the size of `get(M,'eqtnAlias')`.

Descriptions and aliases of variables, shocks, and parameters

- `'yDescript='`, `'xDescript='`, `'eDescript='`, `'pDescript='` [cellstr] - Change the descriptions of, respectively, measurement variables (y), transition variables (x), shocks (e), and exogenous variables (g).
- `'descript='` [struct] - Change the descriptions of all variables, parameters, and shocks; needs to be a struct (database) with fields corresponding to model names.
- `'yAlias='`, `'xAlias='`, `'eAlias='`, `'pAlias='` [cellstr] - Change the aliases of, respectively, measurement variables (y), transition variables (x), shocks (e), and exogenous variables (g).
- `'alias='` [struct] - Change the aliases of all variables, parameters, and shocks; needs to be a struct (database) with fields corresponding to model names.

Other requests

- `'nAlt='` [numeric] - Change the number of alternative parameterisations.
- `'stdVec='` [numeric] - Change the whole vector of std deviations.
- `'tOrigin='` [numeric] - Change the base year for computing deterministic time trends in measurement variables.
- `'epsilon='` [numeric] - Change the relative differentiation step when computing Taylor expansion.

---

shockdb

Create model-specific database with random shocks

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
D = shockdb(M,~D,Range,~NDraw,...)
```

Input arguments

- `M` [model] - Model object.

- D [struct | empty] - Input database to which shock time series will be added; if omitted or empty, a new database will be created; if D already contains shock time series, the data generated by shockdb will be added up with the existing data.
- Range [numeric] - Date range on which the shock time series will be generated and returned; if D already contains shock time series going before or after Range, these will be clipped down to Range in the output database.
- NDraw [numeric] - Number of draws (i.e. columns) generated for each shock; if omitted, the number of draws is equal to the number of alternative parameterizations in the model M, or to the number of columns in shock series existing in the input database D.

Output arguments

- D [struct] - Database with shock time series added.

Options

- 'randFunc=' [@lhsnorm | @randn | @zeros] - Function used to generate random draws for new shock time series; if @zeros, the new shocks will simply be filled with zeros; the random numbers will be adjusted by the respective covariance matrix implied by the current model parameterization.%

Description

Example

---

shockplot

Short-cut for running and plotting plain shock simulation

Syntax

```
[S,FF,AA] = shockplot(M,ShockName,SimRange,PlotList,...)
```

Input arguments

- M [model] - Model object that will be simulated.

- ShkName [char] - Name of the shock that will be simulated.
- Range [numeric | char] - Date range on which the shock will be simulated.
- PlotList [cellstr] - List of variables that will be reported; you can use the syntax of dbase/dbplot P408.

Output arguments

- S [struct] - Database with simulation results.
- FF [numeric] - Handles of figure windows created.
- AA [numeric] - Handles of axes objects created.

Options affecting the simulation

- 'deviation=' [true | false] - See the option 'deviation=' in model/simulate P137.
- 'dtrends=' [@auto | true | false] - See the option 'dtrends=' option in model/simulate P137.
- 'shockSize=' ['std' | numeric] - Size of the shock that will be simulated; 'std' means that one std dev of the shock will be simulated.

Options affecting the graphs

See help on dbase/dbplot P408 for other options available.

Description

The simulated shock always occurs at time t=1. Starting the simulation range, SimRange, before t=1 allows you to simulate anticipated shocks.

The graphs automatically include one pre-sample period, i.e. one period prior to the start of the simulation.

Example

■ simulate

Simulate model

Syntax

```
S = simulate(M,D,Range,...)
[S,Flag,AddF,Delta] = simulate(M,D,Range,...)
```

Input arguments

- M [model] - Solved model object.
- D [struct | cell] - Input database or datapack from which the initial conditions and shocks from within the simulation range will be read.
- Range [numeric | char] - Simulation range.

Output arguments

- S [struct | cell] - Database with simulation results.

Output arguments in nonlinear simulations

- ExitFlag [cell | empty] - Cell array with exit flags for nonlinearised simulations.
- AddF [cell | empty] - Cell array of tseries with final add-factors added to first-order approximate equations to make nonlinear equations hold.
- Delta [cell | empty] - Cell array of tseries with final discrepancies between LHS and RHS in equations marked for nonlinear simulations by a double-equal sign.

Options

- 'anticipate=' [true | false] - If true, real future shocks are anticipated, imaginary are unanticipated; vice versa if false.
- 'contributions=' [true | false] - Decompose the simulated paths into contributions of individual shocks.
- 'dbOverlay=' [true | false | struct] - Use the function dboverlay to combine the simulated output data with the input database, (or a user-supplied database); both the data preceeding the simulation range and after the simulation range are appended.
- 'deviation=' [true | false] - Treat input and output data as deviations from balanced-growth path.
- 'dTrends=' [@auto | true | false] - Add deterministic trends to measurement variables.

- 'ignoreShocks=' [true | false] - Read only initial conditions from input data, and ignore any shocks within the simulation range.
- 'method=' ['firstorder' | 'selective' | 'global'] - Method of running simulations; 'firstorder' means first-order approximate solution (calculated around steady state); 'selective' means equation-selective nonlinear method; 'global' means global nonlinear method (available only in models with no leads).
- 'plan=' [plan] - Specify a simulation plan to swap endogeneity and exogeneity of some variables and shocks temporarily, and/or to simulate some nonlinear equations.
- 'progress=' [true | false] - Display progress bar in the command window.
- 'sparseShocks=' [true | false] - Store anticipated shocks (including endogenized anticipated shocks) in sparse array.

Options for equation-selective nonlinear simulations

- 'solver=' [@qad | @fsolve | @lsqnonlin] - Solution algorithm; see Description.
- 'maxNumelJv=' [numeric | 1e6] - Maximum number of data points (nonlinear plus exogenized) allowed for a nonrecursive algorithm in the nonlinear equation updating step; if exceeded, a recursive (period-by-period) simulation is used to update nonlinear equations instead.
- 'nonlinPer=' [numeric | @all] - Horizon (number of periods from the beginning of the simulation, and from the beginning of each simulation segment) over which nonlinearities will be preserved; the remaining periods will be simulated using first-order approximate solution.

Options for equation-selective nonlinear simulations with @qad solver

- 'addSstate=' [true | false] - Add steady state levels to simulated paths before evaluating nonlinear equations; this option is used only if 'deviation=' true.
- 'display=' [true | false | numeric | Inf] - Report iterations on the screen; if 'display=' N, report every N iterations; if 'display=' Inf, report only final iteration.
- 'error=' [true | false] - Throw an error whenever a nonlinear simulation fails converge; if false, only an warning will display.
- 'lambda=' [numeric | 1] - Initial step size (between 0 and 1) for add factors added to nonlinearised equations in every iteration; see also 'nOptimLambda='.
- 'nOptimLambda=' [numeric | false | 1] - Find the optimal step size on a grid of 10 points between 0 and 'lambda=' before each of the first 'nOptimLambda=' iterations; if false, the value assigned to Lambda is used and no grid search is performed.

- 'reduceLambda=' [numeric | 0.5] - Reduction factor (between 0 and 1) by which lambda will be multiplied if the nonlinear simulation gets on an divergence path.
- 'upperBound=' [numeric | 1.5] - Multiple of all-iteration minimum achieved that triggers a reversion to that iteration and a reduciton in lambda.
- 'maxIter=' [numeric | 100] - Maximum number of iterations.
- 'tolerance=' [numeric | 1e-5] - Convergence tolerance.

Options for nonlinear simulations with Optim Tbx solver

- 'optimSet=' [cell | struct] - Optimization Tbx options.

Options for global nonlinear simulations

- 'optimSet=' [cell | struct] - Optimization Tbx options.
- 'solver=' [@fsolve | @lsqnonlin] - Solution algorithm; see Description.

Description

The function simulate(...) simulates a model on the specified simulation range. By default, the simulation is based on a first-order approximate solution (calculated around steady state). To run nonlinear simulations, use the option 'nonlinear=' (to set the number of periods

Output range

Time series in the output database, S, are are defined on the simulation range, Range, plus include all necessary initial conditions, ie. lags of variables that occur in the model code. You can use the option 'dboverlay=' to combine the output database with the input database (ie. to include a longer history of data in the simulated series).

Deviations from steady-state and deterministic trends

By default, both the input database, D, and the output database, S, are in full levels and the simulated paths for measurement variables include the effect of deterministic trends, including possibly exogenous variables. The default behavior can be changed by changing the options 'deviation=' and 'dTrends='.

The default value for 'deviation=' is false. If set to true, then the input database is expected to contain data in the form of deviations from their steady state levels or paths. For ordinary variables (ie. variables whose log status is false), it is $x_t - \bar{x}_t$, meaning that a 0 indicates that the variable is at its steady state and e.g. 2 indicates the variables exceeds its steady state by 2. For log variables

(ie. variables whose log status is true), it is $x_t/\bar{x}_t$, meaning that a 1 indicates that the variable is at its steady state and e.g. 1.05 indicates that the variable is 5 per cent above its steady state.

The default value for 'dTrends=' is @auto. This means that its behavior depends on the option 'deviation='. If 'deviation=' false then deterministic trends are added to measurement variables, unless you manually override this behavior by setting 'dTrends=' false. On the other hand, if 'deviation=' true then deterministic trends are not added to measurement variables, unless you manually override this behavior by setting 'dTrends=' true.

Simulating contributions of shocks

Use the option 'contributions=' true to request the contributions of shocks to the simulated path for each variable; this option cannot be used in models with multiple alternative parameterizations or with multiple input data sets.

The output database, S, contains Ne+2 columns for each variable, where Ne is the number of shocks in the model:

- the first columns 1...Ne are the contributions of the Ne individual shocks to the respective variable;
- column Ne+1 is the contribution of initial condition, th econstant, and deterministic trends, including possibly exogenous variables;
- column Ne+2 is the contribution of nonlinearities in nonlinear simulations (it is always zero otherwise).

The contributions are additive for ordinary variables (ie. variables whose log status is false), and multplicative for log variables (ie. variables whose log status is true). In other words, if S is the output database from a simulation with 'contributions=' true, X is an ordinary variable, and Z is a log variable, then

```
sum(S.X,2)
```

(ie. the sum of all Ne+2 contributions in each period, ie. summation goes across 2nd dimension) reproduces the final simulated path for the variable X, whereas

```
prod(S.Z,2)
```

(ie. the product of all Ne+2 contributions) reproduces the final simulated path for the variable Z.

Simulations with multiple parameterisations and/or multiple data sets

If you simulate a model with N parameterisations and the input database contains K data sets (ie. each variable is a time series with K columns), then the following happens:

- The model will be simulated a total of P = max(N,K) number of times. This means that each variables in the output database will have P columns.
- The 1st parameterisation will be simulated using the 1st data set, the 2nd parameterisation will be simulated using the 2nd data set, etc. until you reach either the last parameterisation or the last data set, ie. min(N,K). From that point on, the last parameterisation or the last data set will be simply repeated (re-used) in the remaining simulations.
- Formally, the I-th column in the output database, where I = 1, ..., P, is a simulation of the min(I,N)-th model parameterisation using the min(I,K)-th input data set number.

Equation-selective nonlinear simulations

The equation-selective nonlinear simulation approach is invoked by setting 'method=' 'selective'. In equation-selective nonlinear simulations, the solver tries to find add-factors to user-selected nonlinear equations (ie. equations with =# instead of the equal sign in the model file) in the first-order solution such that the original nonlinear equations hold for simulated trajectories (with expectations replaced with actual leads).

Two numerical approaches are available, controlled by the option 'solver=':

- 'QaD' - a quick-and-dirty, but less robust method (default);
- @fsolve, @lsqnonlin - which are standard Optimization Tbx routines, slower but likely to converge for a wider variety of simulations.

Global nonlinear simulations

The global nonlinear simulation approach is invoked by setting 'method=' 'global' and is available only in models with no leads (expectations). In global nonlinear simulations, the entire model is solved as a system of nonlinear equations, period by period, using one of the following two Optimization Tbx routines: @fsolve or @lsqnonlin (default).

Example

■ single

Convert solution matrices to single precision

Syntax

```
m = single(m)
```

Input arguments

- m [model] - Model objects whose solution matrices will be converted to single precision.

Output arguments

- m [model] - Model objects single-precision solution matrices.

Description

---

solve

Calculate first-order accurate solution of the model

Syntax

```
M = solve(M,...)
```

Input arguments

- M [model] - Paramterised model object. Non-linear models must also have a steady state values assigned.

Output arguments

- M [model] - Model with newly computed solution.

Options

- 'expand=' [numeric | 0 | NaN] - Number of periods ahead up to which the model solution will be expanded; if NaN the matrices needed to support solution expansion are not calculated and stored at all and the model cannot be used later in simulations or forecasts with anticipated shocks or plans.

- 'eqtn=' [all | 'measurement' | 'transition'] - Update existing solution in the measurement block, or the transition block, or both.
- 'error=' [true | false] - Throw an error if no unique stable solution exists; if false, a warning message only will be displayed.
- 'linear=' [@auto | true | false] - Solve the model using a linear approach, i.e. differentiating around zero and not the currently assigned steady state.
- 'progress=' [true | false] - Display progress bar in the command window.
- 'select=' [true | false] - Automatically detect which equations need to be re-differentiated based on parameter changes from the last time the system matrices were calculated.
- 'warning=' [true | false] - Display warnings produced by this function.

Description

The IRIS solver uses an ordered QZ (or generalised Schur) decomposition to integrate out future expectations. The QZ may (very rarely) fail for numerical reasons. IRIS includes two patches to handle the some of the QZ failures: a SEVN2 patch (Sum-of-EigenValues-Near-Two), and an E2C2S patch (Eigenvalues-Too-Close-To-Swap).

- The SEVN2 patch: The model contains two or more unit roots, and the QZ algorithm interprets some of them incorrectly as pairs of eigenvalues that sum up accurately to 2, but with one of them significantly below 1 and the other significantly above 1. IRIS replaces the entries on the diagonal of one of the QZ factor matrices with numbers that evaluate to two unit roots.
- The E2C2S patch: The re-ordering of thq QZ matrices fails with a warning 'Reordering failed because some eigenvalues are too close to swap.' IRIS attempts to re-order the equations until QZ works. The number of attempts is limited to N-1 at most where N is the total number of equations.

Example

■ srf

Shock response functions, first-order solution only

Syntax

```
S = srf(M,NPer,...)
S = srf(M,Range,...)
```

Input arguments

- `M` [model] - Model object whose shock responses will be simulated.
- `Range` [numeric | char] - Simulation date range with the first date being the shock date.
- `NPer` [numeric] - Number of simulation periods.

Output arguments

- `S` [struct] - Database with shock response time series.

Options

- `'delog='` [`true` | `false`] - Delogarithmise shock responses for log variables afterwards.
- `'select='` [cellstr | `@all`] - Run the shock response function for a selection of shocks only; `@all` means all shocks are simulated.
- `'size='` [`@auto` | numeric] - Size of the shocks that will be simulated; `@auto` means that each shock will be set to its std dev currently assigned in the model object `M`.

Description

Example

---

sspace

State-space matrices describing the model solution

Syntax

```
[T,R,K,Z,H,D,U,Omg] = sspace(M,...)
```

Input arguments

- `M` [model] - Solved model object.

Output arguments

- `T` [numeric] - Transition matrix.
- `R` [numeric] - Matrix at the shock vector in transition equations.
- `K` [numeric] - Constant vector in transition equations.
- `Z` [numeric] - Matrix mapping transition variables to measurement variables.
- `H` [numeric] - Matrix at the shock vector in measurement equations.
- `D` [numeric] - Constant vector in measurement equations.
- `U` [numeric] - Transformation matrix for predetermined variables.
- `Omg` [numeric] - Covariance matrix of shocks.

Options

- `'triangular='` [`true` | `false`] - If true, the state-space form returned has the transition matrix `T` quasi triangular and the vector of predetermined variables transformed accordingly; this is the default form used in IRIS calculations. If false, the state-space system is based on the original vector of transition variables.

Description

The state-space representation has the following form:

```
[xf;alpha] = T*alpha(-1) + K + R*e

y = Z*alpha + D + H*e

xb = U*alpha

Cov[e] = Omg
```

where `xb` is an nb-by-1 vector of predetermined (backward-looking) transition variables and their auxiliary lags, `xf` is an nf-by-1 vector of non-predetermined (forward-looking) variables and their auxiliary leads, `alpha` is a transformation of `xb`, `e` is an ne-by-1 vector of shocks, and `y` is an ny-by-1

vector of measurement variables. Furthermore, we denote the total number of transition variables, and their auxiliary lags and leads, nx = nb + nf.

The transition matrix, `T`, is, in general, rectangular nx-by-nb. Furthremore, the transformed state vector alpha is chosen so that the lower nb-by-nb part of `T` is quasi upper triangular.

You can use the `get(m,'xVector')` function to learn about the order of appearance of transition variables and their auxiliary lags and leads in the vectors `xb` and `xf`. The first nf names are the vector `xf`, the remaining nb names are the vector `xb`.

■ sstate

Compute steady state or balance-growth path of the model

Syntax

```
[M,Flag] = sstate(M,...)
```

Input arguments

- `M` [model] - Parameterised model object.

Output arguments

- `M` [model] - Model object with newly computed steady state assigned.
- `Flag` [`true` | `false`] - True for parameterizations where steady state has been found successfully.

Options

- `'linear='` [`@auto` | `true` | `false`] - Solve for steady state using a linear approach, i.e. based on the first-order solution matrices and the vector of constants.
- `'warning='` [`true` | `false`] - Display IRIS warning produced by this function.

Options for nonlinear models

- `'blocks='` [`true` | `false`] - Re-arrarnge steady-state equations in recursive blocks before computing steady state.

- 'display=' ['iter' | 'final' | 'notify' | 'off'] - Level of screen output, see Optim Tbx.
- 'endogenise=' [cellstr | char | empty] - List of parameters that will be endogenised when computing the steady state; the number of endogenised parameters must match the number of transtion variables exogenised in the 'exogenised=' option.
- 'exogenise=' [cellstr | char | empty] - List of transition variables that will be exogenised when computing the steady state; the number of exogenised variables must match the number of parameters exogenised in the 'exogenise=' option.
- 'fix=' [cellstr | empty] - List of variables whose steady state will not be computed and kept fixed to the currently assigned values.
- 'fixAllBut=' [cellstr | empty] - Inverse list of variables whose steady state will not be computed and kept fixed to the currently assigned values.
- 'fixGrowth=' [cellstr | empty] - List of variables whose steady-state growth will not be computed and kept fixed to the currently assigned values.
- 'fixGrowthAllBut=' [cellstr | empty] - Inverse list of variables whose steady-state growth will not be computed and kept fixed to the currently assigned values.
- 'fixLevel=' [cellstr | empty] - List of variables whose steady-state levels will not be computed and kept fixed to the currently assigned values.
- 'fixLevelAllBut=' [cellstr | empty] - Inverse list of variables whose steady-state levels will not be computed and kept fixed to the currently assigned values.
- 'growth=' [true | false] - If true, both the steady-state levels and growth rates will be computed; if false, only the levels will be computed assuming that the model is either stationary or that the correct steady-state growth rates are already assigned in the model object.
- 'logMinus=' [cell | char | empty] - List of log variables whose steady state will be restricted to negative values in this run of sstate.
- 'optimSet=' [cell | empty] - Name-value pairs with Optim Tbx settings; see help optimset for details on these settings.
- 'reuse=' [true | false] - Reuse the steady-state values calculated for a parameterisation to initialise the next parameterisation.
- 'solver=' ['fsolve' | 'lsqnonlin'] - Numerical routine to solve for steady state of nonlinear models; it can be one of the two Optimization Tbx functions.
- 'sstate=' [true | false | cell] - If true or a cell array, the steady state is re-computed in each iteration; the cell array can be used to modify the default options with which the sstate function is called.

- 'unlog=' [cell | char | empty] - List of log variables that will be temporarily treated as non-log variables in this run of sstate, i.e. their steady-state levels will not be restricted to either positive or negative values.

Options for linear models

- 'solve=' [true | false] - Solve model before computing steady state.

Description

Non-stationary models

For backward compatibility, the option 'growth=' is set to false by default so that either the model is assumed stationary or the steady-state growth rates have been already pre-assigned to the model object. To use the sstate function for computing both the steady-state levels and steady-state growth rates in a balanced-growth model, you need to set the option 'growth=' true.

Lower and upper bounds

Use options 'levelBounds=' and 'growthBounds=' to impose lower and/or upper bounds on steady-state levels and/or growth rates of selected variables. Create a struct with a 1-by-2 vector [lowerBnd,upperBnd] for each variable that is supposed to be bounded when the steady state is being calculated, and pass the struct into the respective option. User -Inf or Inf if only one of the bounds is specified. For instance, the following piece of code

```
bnd = struct();
bnd.X = [0,10];
bnd.Y = [-Inf,20];
bnd.Z = [5,Inf];
```

specifies lower bounds for variables X and Z, and upper bounds for variables X and Y. The variables that are not bounded do not need to be included in the struct.

Example

■ sstatedb

Create model-specific steady-state or balanced-growth-path database

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[D,IsDev] = sstatedb(M,Range,~NCol,...)
```

Input arguments

- `M` [model] - Model object for which the sstate database will be created.
- `Range` [numeric] - Intended simulation range; the steady-state or balanced-growth-path database will be created on a range that also automatically includes all the necessary lags.
- `~NCol` [numeric | 1] - Number of columns created in the time series object for each variable; the input argument `NCol` can be only used on models with one parameterisation; if omitted `NCol=1`.

Options

- `'randFunc='` [`@lhsnorm` | `@randn` | `@zeros`] - Function used to generate random draws for shock time series; if `@zeros`, the shocks will simply be filled with zeros; the random numbers will be adjusted by the respective covariance matrix implied by the current model parameterization.

Output arguments

- `D` [struct] - Database with a steady-state or balanced-growth path tseries object for each model variable, and a scalar or vector of the currently assigned values for each model parameter.
- `IsDev` [`false`] - The second output argument is always `false`, and can be used to set the option `'deviation='` in model/simulate P137.

Description

Example

---

stdscale

Rescale all std deviations by the same factor

Syntax

```
This = stdscale(This,Factor)
```

Input arguments

- `This` [model] - Model object whose std deviations will be rescaled.
- `Factor` [numeric] - Factor by which all std deviations in the model object `This` will be rescaled.

Output arguments

- `This` [model] - Model object with all of std deviations rescaled.

Description

Example

---

subsasgn

Subscripted assignment for model and systemfit objects

Syntax for assigning parameterisations from other object

```
M(Inx) = N
```

Syntax for deleting specified parameterisations

```
M(Inx) = []
```

Syntax for assigning parameter values or steady-state values

```
M.Name = X
M(Inx).Name = X
M.Name(Inx) = X
```

Syntax for assigning std deviations or cross-correlations of shocks

```
M.std_Name = X
M.corr_Name1__Name2 = X
```

Note that a double underscore is used to separate the Names of shocks in correlation coefficients.

Input arguments

- `M` [model | systemfit] - Model or systemfit object that will be assigned new parameterisations or new parameter values or new steady-state values.
- `N` [model | systemfit] - Model or systemfit object compatible with `M` whose parameterisations will be assigned (copied) into `M`.
- `Inx` [numeric] - Inx of parameterisations that will be assigned or deleted.
- `Name`, `Name1`, `Name2` [char] - Name of a variable, shock, or parameter.
- `X` [numeric] - A value (or a vector of values) that will be assigned to a parameter or variable Named `Name`.

Output arguments

- `M` [model | systemfit] - Model or systemfit object with newly assigned or deleted parameterisations, or with newly assigned parameters, or steady-state values.

Description

Example

Expand the number of parameterisations in a model or systemfit object that has initially just one parameterisation:

```
m(1:10) = m;
```

The parameterisation is simply copied ten times within the model or systemfit object.

■ subsref

Subscripted reference for model and systemfit objects

Syntax for retrieving object with subset of parameterisations

```
M(Inx)
```

Syntax for retrieving parameters or steady-state values

```
M.Name
```

Syntax to retrieve a std deviation or a cross-correlation of shocks

```
M.std_ShockName
M.corr_ShockName1__ShockName2
```

Note that a double underscore is used to separate the names of shocks in correlation coefficients.

Input arguments

- `M` [model | systemfit] - Model or systemfit object.
- `Inx` [numeric | logical] - Inx of requested parameterisations.
- `Name` - Name of a variable, shock, or parameter.
- `ShockName1`, `ShockName2` - Name of a shock.

Description

Example

■ system

System matrices for unsolved model

Syntax

```
[A,B,C,D,F,G,H,J,List,Nf] = system(M)
```

Input arguments

- M [model] - Model object whose system matrices will be returned.

Output arguments

- A, B, C, D, F, G, H ,J [numeric] - Matrices describing the unsolved system, see Description.
- List [cell] - Lists of measurement variables, transition variables includint their auxiliary lags and leads, and shocks as they appear in the rows and columns of the system matrices.
- Nf [numeric] - Number of non-predetermined (forward-looking) transition variables (multiplied by the first Nf columns of matrices A and B).

Options

- 'linear=' [@auto | true | false] - Compute the model using a linear approach, i.e. differentiating around zero and not the currently assigned steady state.
- 'select=' [true | false] - Automatically detect which equations need to be re-differentiated based on parameter changes from the last time the system matrices were calculated.
- 'sparse=' [true | false] - Return matrices A, B, D, F, G, and J as sparse matrices; can be set to true only in models with one parameterization.

Description

The system before the model is solved has the following form:

```
A E[xf;xb] + B [xf(-1);xb(-1)] + C + D e = 0

F y + G xb + H + J e = 0
```

where E is a conditional expectations operator, xf is a vector of non-predetermined (forward-looking) transition variables, xb is a vector of predetermined (backward-looking) transition variables, y is a vector of measurement variables, and e is a vector of transition and measurement shocks.

Example

templatedb

Create model-specific template database

Syntax

```
D = templatedb(M)
```

Input arguments

- M [model] - Model object for which the empty template database will be created.

Output arguments

- D [struct] - Empty database with a field for each of the model variables, shocks, and parameters.

Description

Example

---

tolerance

Get or set model-specific tolerance levels

Syntax for getting tolerance

```
TolStruct = tolerance(M)
Tol = tolerance(M,Scope)
```

Syntax for setting tolerance

```
M = tolerance(M,TolStruct)
M = tolerance(M,@default)
M = tolerance(M,Scope,Tol)
M = tolerance(M,Scope,@default)
```

Input arguments

- `M` [model] - Model object.
- `Scope` [`'solve'` | `'steady'` | `'eigen'` | `'mse'`] - Scope in which the new tolerance level will be used.
- `Tol` [numeric] - New tolerance level used to detect singularities and unit roots; if `@default` tolerance will be set to its default value.
- `TolStruct` [numeric] - Structure with new levels of tolerance for each scope.

Output arguments

- `Tol` [numeric] - Currently assigned level of tolerance.
- `TolStruct` [numeric] - Structure with currently assigned levels of tolerance for each scope.
- `M` [model] - Model object with the new level of tolerance set.

Description

---

unlock

Unlock (enable) locked dynamic links or sstate update equations

Syntax

```
M = unlock(M,'!links')
M = unlock(M,'!links',Name1,Name2,...);
M = unlock(M,'!sstate_update');
M = unlock(M,'!sstate_update',Name1,Name2,...);
```

Input arguments

- `M` [model] - Model object.
- `Name1`, `Name2` [char] - List of names whose links or sstate update equations will be unlocked.

Output arguments

- `M` [model] - Model object with dynamic links `!links` P36 or steady-state update equations `!sstate_update` P44 enabled.

Example

■ userdata

Get or set user data in an IRIS object

Syntax for getting user data

```
X = userdata(Obj)
```

Syntax for assigning user data

```
OBJ = userdata(Obj,X)
```

Input arguments

- `Obj` [model | tseries | VAR | SVAR | FAVAR] - One of the IRIS objects with access to user data functions.
- `X` [...] - Any kind of data that will be attached to, and stored within, the object `OBJ`.

Output arguments

- `X` [...] - User data that are currently attached to the object.
- `Obj` [model | tseries | VAR | SVAR | FAVAR] - The object with its user data updated.

Description

Example

VAR

Population VAR for selected model variables

Syntax

```
V = VAR(M,List,Range,...)
```

Input arguments

- `M` [model] - Solved model object.
- `List` [cellstr | char] - List of variables selected for the VAR.
- `Range` [numeric | char] - Hypothetical range, including pre-sample initial condition, on which the VAR would be estimated.

Output arguments

- `V` [VAR] - Asymptotic reduced-form VAR for selected model variables.

Options

- `'order='` [numeric | 1] - Order of the VAR.
- `'constant='` [true | `false`] - Include in the VAR a constant vector derived from the steady state of the selected variables.

Description

Example

---

vma

Vector moving average representation of the model

Syntax

```
[Phi,List] = vma(M,P,...)
```

Input arguments

- `M` [model] - Model object for which the VMA representation will be computed.
- `P` [numeric] - Order up to which the VMA will be evaluated.

Output arguments

- `Phi` [namedmat | numeric] - VMA matrices.
- `List` [cell] - List of measurement and transition variables in the rows of the `Phi` matrix, and list of shocks in the columns of the `Phi` matrix.

Option

- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrix `Phi` as either a `namedmat` P204 object (i.e. matrix with named rows and columns) or a plain numeric array.
- `'select='` [`@all` | char | cellstr] - Return VMA for selected variables only; `@all` means all variables.

Description

Example

---

xsf

Power spectrum and spectral density of model variables

Syntax

```
[S,D,List] = xsf(M,Freq,...)
[S,D,List,Freq] = xsf(M,NFreq,...)
```

Input arguments

- `M` [model] - Model object.
- `Freq` [numeric] - Vector of frequencies at which the XSFs will be evaluated.

- `NFreq` [numeric] - Total number of requested frequencies; the frequencies will be evenly spread between 0 and `pi`.

Output arguments

- `S` [namedmat | numeric] - Power spectrum matrices.
- `D` [namedmat | numeric] - Spectral density matrices.
- `List` [cellstr] - List of variable in order of appearance in rows and columns of `S` and `D`.
- `Freq` [numeric] - Vector of frequencies at which the XSFs has been evaluated.

Options

- `'applyTo='` [cellstr | char | `@all`] - List of variables to which the option `'filter='` will be applied; `@all` means all variables.
- `'filter='` [char | empty] - Linear filter that is applied to variables specified by 'applyto'.
- `'nFreq='` [numeric | `256`] - Number of equally spaced frequencies over which the 'filter' is numerically integrated.
- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrices `S` and `D` as either `namedmat` P204 objects (i.e. matrices with named rows and columns) or plain numeric arrays.
- `'progress='` [`true` | `false`] - Display progress bar on in the command window.
- `'select='` [`@all` | char | cellstr] - Return XSF for selected variables only; `@all` means all variables.

Description

Example

■ zerodb

Create model-specific zero-deviation database

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[D,IsDev] = zerodb(M,Range,~NCol,...)
```

Input arguments

- `M` [model] - Model object for which the zero database will be created.
- `Range` [numeric] - Intended simulation range; the zero database will be created on a range that also automatically includes all the necessary lags.
- `~NCol` [numeric | 1] - Number of columns created in the time series object for each variable; the input argument `NCol` can be only used on models with one parameterisation; may be omitted.

Option

- `'randFunc='` [`@lhsnorm` | `@randn` | `@zeros`] - Function used to generate random draws for shock time series; if `@zeros`, the shocks will simply be filled with zeros; the random numbers will be adjusted by the respective covariance matrix implied by the current model parameterization.

Output arguments

- `D` [struct] - Database with a tseries object filled with zeros for each linearised variable, a tseries object filled with ones for each log-linearised variables, and a scalar or vector of the currently assigned values for each model parameter.
- `IsDev` [`true`] - The second output argument is always `true`, and can be used to set the option `'deviation='` in `model/simulate` P137.

Description

Example

6 Reporting Equations (rpteq Objects)

Reporting equations (rpteq) objects are systems of equations evaluated successively (i.e. not simultaneously) equation by equation, period by period.

There are three basic ways to create reporting equations objects:

- in the !reporting_equations P43 section of a model file;
- in a separate reporting equations file;
- on the fly within an m-file or in the command window.

Rpteq methods:

Constructor

- rpteq P163 - New reporting equations (rpteq) object.

Evaluating reporting equations

- run P164 - Evaluate reporting equations (rpteq) object.

Evaluating reporting equations from within model object

- reporting P129 - Evaluate reporting equations from within model object.

Getting on-line help on rpteq functions

```
help rpteq
help rpteq/function_name
```

Reference page for rpteq

■ reporting

Evaluate reporting equations from within model object

Syntax

```
D = reporting(M,D,Range,...)
```

Input arguments

- M [model] - Model object with reporting equations.
- D [struct] - Input database that will be used to evaluate the reporting equations.
- Range [numeric | char] - Date range on which the reporting equations will be evaluated.

Output arguments

- D [struct] - Output database with reporting variables.

Options

See rpteq/run P164 for options available.

Description

---

rpteq

New reporting equations (rpteq) object

Syntax

```
Q = rpteq(FName)
Q = rpteq(Eqtn)
```

Input arguments

- FName [char | cellstr] - File name or cellstr array of file names, each a plain text file with reporting equations; multiple input files will be combined together.
- Eqtn [char | cellstr] - Equation or cellstr array of equations.

Output arguments

- `Q` [rpteq] - New reporting equations object.

Description

Reporting equations must be written in the following form:

```
`LhsName = RhsExpr;`
`"Label" LhsName = RhsExpr;`
```

where

- `LhsName` is the name of a left-hand-side variable (with no lag or lead);
- `RhsExpr` is an expression on the right-hand side that will be evaluated period by period, and assigned to the left-hand-side variable, `LhsName`.
- `"Label"` is an optional label that will be used to create a comment in the output time series for the respective left-hand-side variable.
- the equation must end with a semicolon.

Example

```
q = rpteq({ ...
    'a = c * a{-1}^0.8 * b{-1}^0.2;', ...
    'b = sqrt(b{-1});', ...
    })

q =
    rpteq object
    number of equations: [2]
    comment: ''
    user data: empty
    export files: [0]
```

---

run

Evaluate reporting equations (rpteq) object

Syntax

```
Outp = run(Q,Inp,Range,...)
```

Input arguments

- `Q` [char] - Reporting equations (rpteq) object.
- `Inp` [struct] - Input database that will be used to evaluate the reporting equations.
- `Dates` [numeric | char] - Dates at which the reporting equations will be evaluated; `Dates` does not need to be a continuous date range.

Output arguments

- `Outp` [struct] - Output database with reporting variables.

Options

- `'dbOverlay='` [`true` | `false` | struct] - If `true`, the LHS output data will be combined with data from the input database (or a user-supplied database).
- `'fresh='` [`true` | `false`] - If `true`, only the LHS reporting variables will be included in the output database, `Outp`; if `false` the output database will also include all entries from the input database, `Inp`.

Description

Reporting equations are always evaluated non-simultaneously, i.e. equation by equation, for each period.

Example

Note the differences in the three output databases, `d1`, `d2`, `d3`, depending on the options `'dbOverlay='` and `'fresh='`.

```
>> q = rpteq({ ...
    'a = c * a{-1}^0.8 * b{-1}^0.2;', ...
    'b = sqrt(b{-1});', ...
    })
```

```
q =
    rpteq object
    number of equations: [2]
    comment: ''
    user data: empty
    export files: [0]

>> d = struct();
>> d.a = tseries();
>> d.b = tseries();
>> d.a(qq(2009,4)) = 0.76;
>> d.b(qq(2009,4)) = 0.88;
>> d.c = 10;
>> d

d =
    a: [1x1 tseries]
    b: [1x1 tseries]
    c: 10

>> d1 = run(q,d,qq(2010,1):qq(2011,4))

d1 =
    a: [8x1 tseries]
    b: [8x1 tseries]
    c: 10

>> d2 = run(q,d,qq(2010,1):qq(2011,4),'dbOverlay=',true)

d2 =
    a: [9x1 tseries]
    b: [9x1 tseries]
    c: 10

>> d3 = run(q,d,qq(2010,1):qq(2011,4),'fresh=',true)

d3 =
    a: [8x1 tseries]
    b: [8x1 tseries]
```

7 Model Simulation Plans (plan Objects)

Simulation plans complement the use of the model/simulate P137 or model/jforecast P115 functions.

You need to use a simulation plan object to set up the following types of more complex simulations or forecasts (or a combination of these):

- simulations or forecasts with some of the model variables temporarily exogenised;
- simulations with some of the non-linear equations solved in an exact non-linear mode;
- forecasts conditioned upon some variables;

The plan object is passed to the model/simulate P137 or model/jforecast P115 functions through the 'plan=' option.

Plan methods:

Constructor

- plan P175 - Create new empty simulation plan object.

Getting information about simulation plans

- detail P169 - Display details of a simulation plan.
- get P172 - Query to plan object.
- nnzcond P173 - Number of conditioning data points.
- nnzendog P174 - Number of endogenised data points.
- nnzexog P174 - Number of exogenised data points.

Setting up simulation plans

- autoexogenise P168 - Exogenise variables and automatically endogenise corresponding shocks.
- condition P169 - Condition forecast upon the specified variables at the specified dates.
- endogenise P170 - Endogenise shocks or re-endogenise variables at the specified dates.
- exogenise P171 - Exogenise variables or re-exogenise shocks at the specified dates.
- reset P171 - Remove all endogenized, exogenized, autoexogenized and conditioned upon data points from simulation plan.
- swap P177 - Swap endogeneity and exogeneity of variables and shocks.

Referencing plan objects

- subsref P176 - Subscripted reference for plan objects.

Getting on-line help on simulation plans

```
help plan
help plan/function_name
```

Reference page for plan

■ autoexogenise

Exogenise variables and automatically endogenise corresponding shocks

Syntax

```
P = autoexogenise(P,List,Dates)
P = autoexogenise(P,List,Dates,Sigma)
```

Input arguments

- `P` [plan] - Simulation plan.
- `List` [cellstr | char | `@all`] - List of variables that will be exogenised; these variables must have their corresponding shocks assigned, see `!autoexogenise` P26; `@all` means all autoexogenised variables defined in the model object will be exogenised.
- `Dates` [numeric] - Dates at which the variables will be exogenised.
- `Sigma` [`1` | `1i` | numeric] - Anticipation mode (real or imaginary) for endogenized shocks, and their numerical weight (used in underdetermined simulation plans); if omitted, `Sigma = 1`.

Output arguments

- `P` [plan] - Simulation plan with new information on exogenised variables and endogenised shocks included.

Description

Example

---

condition

Condition forecast upon the specified variables at the specified dates

Syntax

```
P = condition(P,List,Dates)
```

Input arguments

- P [plan] - Simulation plan.
- List [cellstr | char] - List of variables upon which a forecast will be conditioned.
- Dates [numeric] - Dates at which the forecast will be conditioned upon the specified variables.

Output arguments

- P [plan] - Simulation plan with new conditioning information included.

Description

Example

---

detail

Display details of a simulation plan

Syntax

```
detail(P)
detail(P,Data)
```

Input arguments

- P [plan] - Simulation plan.
- Data [struct] - Input database.

Description

If you supply also the second input argument, the input database `D`, both the dates and the respective values will be reported for exogenised and conditioning data points, and the values will be checked for the presence of NaNs (with a warning should there be found any).

Example

---

endogenise

Endogenise shocks or re-endogenise variables at the specified dates

Syntax

```
P = endogenise(P,List,Dates)
P = endogenise(P,Dates,List)
P = endogenise(P,List,Dates,Sigma)
P = endogenise(P,Dates,List,Sigma)
```

Input arguments

- `P` [plan] - Simulation plan.
- `List` [cellstr | char] - List of shocks that will be endogenised, or list of variables that will be re-endogenise.
- `Dates` [numeric | `@all`] - Dates at which the shocks or variables will be endogenised; `@all` means the entire simulation range specified when creating the plan object.
- `Sigma` [`1` | `1i` | numeric] - Anticipation mode (real or imaginary) for endogenized shocks, and their numerical weight (used in underdetermined simulation plans); if omitted, `Sigma = 1`.

Output arguments

- `P` [plan] - Simulation plan with new information on endogenised shocks included.

Description

Example

---

endogenise

Remove all endogenized, exogenized, autoexogenized and conditioned upon data points from simulation plan

Syntax

```
P = reset(P)
```

Input arguments

- `P` [plan] - Simulation plan.

Output arguments

- `P` [plan] - Simulation plan with all endogenized, exogenized, autoexogenized and conditioned upon data points removed.

Description

Example

---

exogenise

Exogenise variables or re-exogenise shocks at the specified dates

Syntax

```
P = exogenise(P,List,Dates)
P = exogenise(P,Dates,List)
P = exogenise(P,List,Dates,Sigma)
P = exogenise(P,Dates,List,Sigma)
```

Input arguments

- `P` [plan] - Simulation plan.
- `List` [cellstr | char] - List of variables that will be exogenised, or list of shocks that will be re-exogenised.
- `Dates` [numeric | @all] - Dates at which the variables will be exogenised; `@all` means the entire simulation range specified when creating the plan object.
- `Sigma` [`1` | `1i`] - Only when re-exogenising shocks: Select the anticipation mode in which the shock will be re-exogenised; if omitted, `Sigma = 1`.

Output arguments

- `P` [plan] - Simulation plan with new information on exogenised variables included.

Description

Example

■ get

Query to plan object

Syntax

```
Ans = get(P,Query)
[Ans,Ans,...] = get(P,Query,Query,...)
```

Input arguments

- `P` [plan] - Simulation plan object.
- `Query` [char] - Name of the queried property.

Output arguments

- `Ans` [...] - Answer.

Valid queries to plan objects

- `'endogenised='` – Returns [struct] a database with time series for each shock with 1 in each period where the variable is endogenised, and 0 in each period where the variable is not endogenised.
- `'exogenised='` – Returns [struct] a database with time series for each measurement and transition variable with 1 in each period where the variable is exogenised, and 0 in each period where the variable is not exogenised.
- `'onlyEndogenised='` – Returns [struct] the same database as `'endogenised='` but including only those shocks that are endogenised at least in one period.
- `'onlyExogenised='` – Returns [struct] the same database as `'exogenised='` but including only those measurement and transition variables that are endogenised at least in one period.
- `'range='` – Returns [numeric] the simulation plan range.

Description

Example

---

nnzcond

Number of conditioning data points

Syntax

```
N = nnzcond(P)
```

Input arguments

- `P` [plan] - Simulation plan.

Output arguments

- `N` [numeric] - Number of conditioning data points; each variable at each date counts as one data point.

Description

Example

---

nnzendog

Number of endogenised data points

Syntax

```
[N,NReal,NImag] = nnzendog(P)
```

Input arguments

- `P` [plan] - Simulation plan.

Output arguments

- `N` [numeric] - Total number of endogenised data points; each shock at each time counts as one data point.
- `NRea,` [numeric] - Number of endogenised data points with anticipation mode 1.
- `NImag` [numeric] - Number of endogenised data points with anticipation mode 1i.

Description

Example

---

nnzexog

Number of exogenised data points

Syntax

```
N = nnzexog(P)
```

Input arguments

- P [plan] - Simulation plan.

Output arguments

- N [numeric] - Number of exogenised data points; each variable at each date counts as one data point.

Description

Example

---

plan

Create new empty simulation plan object

Syntax

```
P = plan(M,Range)
```

Input arguments

- M [model] - Model object that will be simulated subject to this simulation plan.
- Range [numeric | char] - Simulation range; this range must exactly correspond to the range on which the model will be simulated.

Output arguments

- P [plan] - New empty simulation plan.

Description

You need to use a simulation plan object to set up the following types of more complex simulations or forecats:

- simulations or forecasts with some of the model variables temporarily exogenised;

- simulations with some of the non-linear equations solved exactly.
- forecasts conditioned upon some variables;

The plan object is passed to the simulate P137 or jforecast P115 functions through the option 'plan='.

Example

---

subsref

Subscripted reference for plan objects

Syntax

```
P = P(StartDate:EndDate)
P = P{Shift}
```

Input arguments

- P [plan] - Simulation plan.

Output aguments

- P [plan] - Simulation plan reduced, expanded, or shifted to the new range,
- StartDate [numeric] - New start date for the simulation plan.
- EndDate [numeric] - New end date for the simulation plan.
- Shift [numeric] - Lag or lead by which the simulation plan range will be shifted.

Description

Example

---

swap

Swap endogeneity and exogeneity of variables and shocks

Syntax

```
P = swap(P,ExogList,EndogList,Dates)
P = swap(P,ExogList,EndogList,Dates,Sigma)
```

Input arguments

- `P` [plan] - Simulation plan.
- `ExogList` [cellstr | char] - List of variables that will be exogenized.
- `EndogList` [cellstr | char] - List of shocks that will be endogenized.
- `Dates` [numeric] - Dates at which the variables and shocks will be exogenized/endogenized.
- `Sigma` [numeric] - Anticipation mode (real or imaginary) for the endogenized shocks, and their numerical weight (used in underdetermined simulation plans); if omitted, `Sigma = 1`.

Output arguments

- `P` [plan] - Simulation plan with new information on exogenized variables and endogenized shocks included.

Description

The function swap is equivalent to the following separate calls to functions exogenize and endogenize:

```
p = exogenize(p,ExogList,Dates);
p = endogenize(p,EndogList,Dates);
```

or

```
p = exogenize(p,ExogList,Dates);
p = endogenize(p,EndogList,Dates,Sigma);
```

if the input argument `Sigma` is provided.

Example

8 Grouping and Aggregation of Contributions (grouping Objects)

Grouping objects can be used for aggregating the contributions of shocks in model simulations, model/simulate P137, or aggregating the contributions of measurement variables in Kalman filtering, model/filter P93.

Grouping methods:

Constructor

- grouping P180 - Create new empty grouping object.

Getting information about groups

- detail P179 - Details of a grouping object.
- isempty P181 - True for empty grouping object.

Setting up and using groups

- addgroup P178 - Add measurement variable group or shock group to grouping object.
- eval P179 - Evaluate contributions in input database S using grouping object G.

Getting on-line help on groups

```
help grouping
help grouping/function_name
```

Reference page for grouping

■ addgroup

Add measurement variable group or shock group to grouping object

Syntax

```
G = addgroup(G,GroupName,GroupContents)
```

Input arguments

- `G` [grouping] - Grouping object.
- `GroupName` [char] - Group name.
- `GroupContents` [char | cell | `Inf`] - Names of shocks or measurement variables to be included in the new group; `GroupContents` can also be regular expressions; `Inf` the group will contain all shocks or measurement variables not included in any existing group.

Output arguments

- `G` [grouping] - Grouping object with the new group.

Description

Example

---

detail

Details of a grouping object

Syntax

```
detail(G)
```

Input arguments

- `G` [grouping] - Grouping object.

Description

Example

---

eval

Evaluate contributions in input database S using grouping object G

Syntax

```
[S,L] = eval(G,S)
```

Input arguments

- G [grouping] - Grouping object.
- S [dbase] - Input dabase with individual contributions.

Output arguments

- S [dbase] - Output database with grouped contributions.
- L [cellstr] - Legend entries based on the list of group names.

Options

- 'append=' [true | false] - Append in the output database all remaining data columns from the input database that do not correspond to any contribution of shocks or measurement variables.

Description

Example

For a model object M, database D and simulation range R,

```
S = simulate(M,D,R,'contributions=',true) ;
G = grouping(M)
...
G = addgroup(G,GroupName,GroupContents) ;
...
S = eval(S,G)
```

■ grouping

Create new empty grouping object

Syntax

```
G = grouping(M,Type)
```

Input arguments

- `M` [model] - Model object.
- `Type` [`'shock'` | `'measurement'`] - Type of grouping object.

Output arguments

- `G` [grouping] - New empty grouping object.

Description

Example

---

isempty

True for empty grouping object

Syntax

```
Flag = isempty(G)
```

Input arguments

- `G` [grouping] - Grouping object.

Output arguments

- `Flag` [`true` | `false`] - True if `G` is an empty grouping object.

Description

Example

```
g = grouping();
isempty(g)
ans =
     1
```

9 System Priors (systempriors Objects)

System priors are priors imposed on the system properties of a model as whole, such as shock response functions, frequency response functions, correlations, or spectral densities; moreover, systempriors objects also allow for priors on combinations of parameters. The system priors can be combined with priors on individual parameters.

Systempriors methods:

Constructor

- systempriors P187 - Create new empty system priors object.

Setting up priors

- prior P185 - Add new prior to system priors object.

Getting information about system priors

- detail P183 - Display details of system priors object.
- isempty P184 - True if system priors object is empty.
- length P184 - Number or priors in system priors object.

Reference page for systempriors

■ detail

Display details of system priors object

Syntax

```
detail(S)
```

Input arguments

- S [systempriors] - System priors, systempriors P183 object.

Description

Example

---

isempty

True if system priors object is empty

Syntax

```
Flag = isempty(S)
```

Input arguments

- S [systempriors] - System priors, systempriors P183, object.

Output arguments

- Flag [true | false] - True if the system priors object, S, is empty, false otherwise.

Description

Example

---

length

Number or priors in system priors object

Syntax

```
N = length(S)
```

Input arguments

- S [systempriors] - System priors, systempriors P183 object.

Output arguments

- N [numeric] - Number of priors imposed in the system priors object, S.

Description

Example

---

prior

Add new prior to system priors object

Syntax

```
S = prior(S,Expr,PriorFn,...)
S = prior(S,Expr,[],...)
```

Input arguments

- S [systempriors] - System priors object.
- Expr [char] - Expression that defines a value for which a prior density will be defined; see Description for system properties that can be referred to in the expression.
- PriorFn [function_handle | empty] - Function handle returning the log of prior density; empty prior function, [], means a uniform prior.

Output arguments

- S [systempriors] - The system priors object with the new prior added.

Options

- 'lowerBound=' [numeric | -Inf] - Lower bound for the prior.
- 'upperBound=' [numeric | Inf] - Upper bound for the prior.

Description

System properties that can be used in Expr

- srf[VarName,ShockName,T] - Plain shock response function of variables VarName to shock ShockName in period T. Mind the square brackets.
- ffrf[VarName,MVarName,Freq] - Filter frequency response function of transition variables TVarName to measurement variable MVarName at frequency Freq. Mind the square brackets.
- corr[VarName1,VarName2,Lag] - Correlation between variable VarName1 and variables VarName2 lagged by Lag periods.
- spd[VarName1,VarName2,Freq] - Spectral density between variables VarName1 and VarName2 at frequency Freq.

If a variable is declared as a log variable [P37], it must be referred to as log(VarName) in the above expressions, and the log of that variables is returned, e.g. srf[log(VarName),ShockName,T]. or ffrf[log(TVarName),MVarName,T].

Expressions involving combinations or functions of parameters

Model parameter names can be referred to in Expr preceded by a dot (period), e.g. .alpha^2 + .beta^2 defines a prior on the sum of squares of the two parameters (alpha and beta).

Example

Create a new empty systemprios object based on an existing model.

```
s = systempriors(m);
```

Add a prior on minus the shock response function of variable ygap to shock eps in period 4. The prior density is lognormal with mean 0.3 and std deviation 0.05;

```
s = prior(s,'-srf[ygap,eps,4]',logdist.lognormal(0.3,0.05));
```

Add a prior on the gain of the frequency response function of transition variable ygap to measurement variable 'y' at frequency 2*pi/40. The prior density is normal with mean 0.5 and std deviation 0.01. This prior says that we wish to keep the cut-off periodicity for trend-cycle decomposition close to 40 periods.

```
s = prior(s,'abs(ffrf[ygap,y,2*pi/40])',logdist.normal(0.5,0.01));
```

Add a prior on the sum of parameters alpha1 and alpha2. The prior is normal with mean 0.9 and std deviation 0.1, but the sum is forced to be between 0 and 1 by imposing lower and upper bounds.

```
s = prior(s,'.alpha1 + .alpha2',logdist.normal(0.9,0.1), ...
    'lowerBound=',0,'upperBound=',1);
```

Add a prior saying that the first 16 periods account for at least 90% of total variability (cyclicality) in a 40-period response of ygap to shock eps. This prior is meant to suppress secondary cycles in shock response functions.

```
s = prior(s, ...
   'sum(abs(srf[ygap,eps,1:16])) / sum(abs(srf[ygap,eps,1:40]))', ...
   [],'lowerBound=',0.9);
```

---

systempriors

Create new empty system priors object

Syntax

```
S = systempriors(M)
```

Input arguments

- M [model] - Model object on whose system properties the priors will be imposed.

Output arguments

- S [systempriors] - New empty system priors object.

Description

Example

10 Posterior Simulator (poster Objects)

Posterior simulator objects allow evaluating the behaviour of the posterior dsitribution, and drawing model parameters from the posterior distibution.

Posterior objects are set up within the model/estimate P85 function and returned as the second output argument - the set up and initialisation of the posterior object is fully automated in this case. Alternatively, you can set up a posterior object manually, by setting all its properties appropriately.

Poster methods:

Constructor

- poster P192 - Create new empty posterior simulation (poster) object.

Evaluating posterior density

- arwm P188 - Adaptive random-walk Metropolis posterior simulator.
- eval P191 - Evaluate posterior density at specified points.
- regen P192 - Regeneration time MCMC Metropolis posterior simulator.

Chain statistics

- stats P193 - Evaluate selected statistics of ARWM chain.

Getting on-line help on model functions

```
help poster
help poster/function_name
```

Reference page for poster

---

arwm

Adaptive random-walk Metropolis posterior simulator

Syntax

```
[Theta,LogPost,ArVec,PosUpd] = arwm(Pos,NDraw,...)
[Theta,LogPost,ArVec,PosUpd,SgmVec,FinalCov] = arwm(Pos,NDraw,...)
```

Input arguments

- Pos [poster] - Initialised posterior simulator object.
- NDraw [numeric] - Length of the chain not including burn-in.

Output arguments

- Theta [numeric] - MCMC chain with individual parameters in rows.
- LogPost [numeric] - Vector of log posterior density (up to a constant) in each draw.
- ArVec [numeric] - Vector of cumulative acceptance ratios.
- PosUpd [poster] - Posterior simulator object with its properties updated so to capture the final state of the simulation.
- SgmVec [numeric] - Vector of proposal scale factors in each draw.
- FinalCov [numeric] - Final proposal covariance matrix; the final covariance matrix of the random walk step is Scale(end)^2*FinalCov.

Options

- 'adaptProposalCov=' [numeric | 0.5] - Speed of adaptation of the Cholesky factor of the proposal covariance matrix towards the target acceptanace ratio, targetAR; zero means no adaptation.
- 'adaptScale=' [numeric | 1] - Speed of adaptation of the scale factor to deviations of acceptance ratios from the target ratio, targetAR.
- 'burnin=' [numeric | 0.10] - Number of burn-in draws entered either as a percentage of total draws (between 0 and 1) or directly as a number (integer greater that one). Burn-in draws will be added to the requested number of draws ndraw and discarded after the posterior simulation.
- 'estTime=' [true | false] - Display and update the estimated time to go in the command window.
- 'firstPrefetch=' [numeric | Inf] - First draw where parallelised pre-fetching will be used; Inf means no pre-fetching.
- 'gamma=' [numeric | 0.8] - The rate of decay at which the scale and/or the proposal covariance will be adapted with each new draw.
- 'initScale=' [numeric | 1/3] - Initial scale factor by which the initial proposal covariance will be multiplied; the initial value will be adapted to achieve the target acceptance ratio.

- 'lastAdapt=' [numeric | Inf] - Last point at which the proposal covariance will be adapted; Inf means adaptation will continue until the last draw. Can also be entered as a percentage of total draws (a number strictly between 0 and 1).
- 'nStep=' [numeric | *1] - Number of pre-fetched steps computed in parallel; only works with firstPrefetch= smaller than NDraw.
- 'progress=' [true | false] - Display progress bar in the command window.
- 'saveAs=' [char | empty] - File name where results will be saved when the option 'saveEvery=' is used.
- 'saveEvery=' [numeric | Inf] - Every N draws will be saved to an HDF5 file, and removed from workspace immediately; no values will be returned in the output arguments Theta, LogPost, AR, Scale; the option 'saveAs=' must be used to specify the file name; Inf means a normal run with no saving.
- 'targetAR=' [numeric | 0.234] - Target acceptance ratio.

Description

The function poster/arwm returns the simulated chain of parameters and the corresponding value of the log posterior density. To obtain simulated sample statistics for each parameter (such as posterior mean, median, percentiles, etc.) use the function poster/stats P193 to process the simulated chain and calculate the statistics.

The properties of the posterior object returned as the 4th output argument are updated so that they capture the final state of the posterior simulations. This can be used to initialize a next simulation at the point where the previous ended.

Parallelised ARWM

Set 'nStep=' greater than 1, and 'firstPrefetch=' smaller than NDraw to start a pre-fetching parallelised algorithm (pre-fetched will be all draws starting from 'firstPrefetch='); to that end, a pool of parallel workers (using e.g. matlabpool from the Parallel Computing Toolbox) must be opened before calling arwm.

With pre-fetching, all possible paths 'nStep=' steps ahead (i.e. all possible combinations of reject/accept) are pre-evaluated in parallel, and then the resulting path is selected. Adapation then occurs only every 'nStep=' steps, and hence the results will always somewhat differ from a serial run. Identical results can be obtained by turning down adaptation before pre-fetching starts, i.e. by setting 'lastAdapt=' smaller than 'firstPrefetch=' (and, obviously, by re-setting the random number generator).

References

- Brockwell, A.E., 2005. "Parallel Markov Chain Monte Carlo Simulation by Pre-Fetching," CMU Statistics Dept. Tech. Report 802.
- Strid, I., 2009. "Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach," SSE/EFI Working Paper Series in Economics and Finance No. 706.

Example

■ eval

Evaluate posterior density at specified points

Syntax

```
[X,L,PP,SrfP,FrfP] = eval(Pos)
[X,L,PP,SrfP,FrfP] = eval(Pos,P)
```

Input arguments

- `Pos` [poster] - Posterior object returned by the `model/estimate` P85 function.
- `P` [struct] - Struct with parameter values at which the posterior density will be evaluated; if `P` is not specified, the posterior density is evaluated at the point of the estimated mode.

Output arguments

- `X` [numeric] - The value of log posterior density evaluated at `P`; N.B. the returned value is log posterior, and not minus log posterior.
- `L` [numeric] - Contribution of data likelihood to log posterior.
- `PP` [numeric] - Contribution of parameter priors to log posterior.
- `SrfP` [numeric] - Contribution of shock response function priors to log posterior.
- `FrfP` [numeric] - Contribution of frequency response function priors to log posterior.

Description

The total log posterior consists, in general, of the four contributions listed above:

```
X = L + PP + SrfP + FrfP.
```

Example

---

poster

Create new empty posterior simulation (poster) object

Syntax

```
P = poster()
```

Description

Creating and initialising posterior simulation objects manually is unnecessary. Posterior simulation objects are created and initialised automatically within estimation methods of various other objects, such as model/estimate P85.

---

regen

Regeneration time MCMC Metropolis posterior simulator

Syntax

```
[Theta,LogPost,AR,Scale,FinalCov] = regen(Pos,NDraw,...)
```

Input arguments

- Pos [poster] - Initialised posterior simulator object.
- NDraw [numeric] - Length of the chain not including burn-in.

Output arguments

- Theta [numeric] - MCMC chain with individual parameters in rows.
- LogPost [numeric] - Vector of log posterior density (up to a constant) in each draw.
- AR [numeric] - Vector of cumulative acceptance ratios in each draw.
- Scale [numeric] - Vector of proposal scale factors in each draw.
- FinalCov [numeric] - Final proposal covariance matrix; the final covariance matrix of the random walk step is Scale(end)^2*FinalCov.

Options

References

- Brockwell, A.E., and Kadane, J.B., 2004. "Identification of Regeneration Times in MCMC Simulation, with Application to Adaptive Schemes," mimeo, Carnegie Mellon University.

Example

---

stats

Evaluate selected statistics of ARWM chain

Syntax

```
S = stats(Pos,Theta,...)
S = stats(Pos,Theta,LogPost,...)
S = stats(Pos,FName,...)
```

Input arguments

- Pos [poster] - Posterior simulator object that has generated the Theta chain.
- Theta [numeric] - MCMC chain generated by the poster/arwm P188 function.
- LogPost [numeric] - Vector of log posterior densities generated by the arwm function; LogPost is not necessary if you do not request 'mdd', the marginal data density.

- FName [char] - File name under which the simulated chain was saved when poster/arwm P188 was run with options saveEvery=' and 'saveAs='.

Output arguments

- S [struct] - Struct with the statistics requested by the user.

Options

- 'estTime=' [true | false] - Display and update the estimated time to go in the command window.
- 'mddGrid=' [numeric | 0.1:0.1:0.9] - Points between 0 and 1 over which the marginal data density estimates will be averaged, see Geweke (1999).
- 'progress=' [true | false] - Display progress bar in the command window.

Options to include/exclude output statistics

- 'bounds=' [true | false] - Include in S the lower and upper parameter bounds set up by the user.
- 'chain=' [true | false] - Include in S the entire simulated chains of parameter values.
- 'cov=' [true | false] - Include in S the sample covariance matrix.
- 'hist=' [numeric | empty] - Include in S histogram bins and counts with the specified number of bins.
- 'hpdi=' [false | numeric] - Include in S the highest probability density intervals with the specified percent coverage (e.g. 90% is entered as 90, not 0.90).
- 'ksdensity=' [true | false | numeric] - Include in S the x- and y-axis points for kernel-smoothed posterior density; use a numeric value to control the number of points over which the density is computed.
- 'mdd=' [true | false] - Include in S minus the log marginal data density.
- 'mean=' [true | false] - Include in S the sample averages.
- 'median=' [true | false] - Include in S the sample medians.
- 'mode=' [true | false] - Include in S the sample modes based on histograms.
- 'prctile=' [numeric | empty] - Include in S the specified percentiles.
- 'std=' [true | false] - Include in S the sample std deviations.

Description

Example

11 Probability Distributions (logdist Package)

The logdist package gives quick access to basic univariate distributions, and in particular to functions proportional to the logarithm of those basic distributions. Its primary use is setting up priors in the model/estimate P85 and poster/arwm P188 functions.

The logdist package is called to create function handles that have several different modes of use. The primary use is to compute values that are proportional to the log of the respective density. In addition, the function handles also give you access to extra information (such as the the proper p.d.f., the name, mean, std dev, mode, and stuctural parameters of the distribution), and to a random number generator from the respective distribution.

Logdist methods:

Getting function handles for univariate distributions

- chisquare P198 - Create function proportional to log of Chi-Squared distribution.
- normal P200 - Create function proportional to log of Normal distribution.
- lognormal P200 - Create function proportional to log of log-normal distribution.
- beta P197 - Create function proportional to log of beta distribution.
- gamma P198 - Create function proportional to log of gamma distribution.
- invgamma P199 - Create function proportional to log of inv-gamma distribution.
- t P201 - Create function proportional to log of Student T distribution.
- uniform P202 - Create function proportional to log of uniform distribution.

Calling the logdist function handles

The function handles F created by the logdist package functions can be called the following ways:

- Get a value proportional to the log-density of the respective distribution at a particular point; this call is used within the posterior simulator P188:

  y = F(x)

- Get the density of the respective distribution at a particular point:

  y = F(x,'pdf')

- Get the characteristics of the distribution – mean, std deviation, mode, and information (the inverse of the second derivative of the log density):

  m = F([],'mean') s = F([],'std') o = F([],'mode') i = F([],'info')

- Get the underlying "structural" parameters of the respective distribution:

  a = F([],'a') b = F([],'b')

- Get the name of the distribution (the names correspond to the function names, i.e. can be either of 'normal', 'lognormal', 'beta', 'gamma', 'invgamma', 'uniform'):

  name = F([],'name')

- Draw a vector or matrix of random numbers from the distribution; drawing from beta, gamma, and inverse gamma requires the Statistics Toolbox:

  a = F([],'draw',1,1000);

  size(a) ans = 1 10000

Getting on-line help on logdist functions

```
help logdist
help logdist/function_name
```

---

beta

Create function proportional to log of beta distribution

Syntax

```
F = logdist.beta(Mean,Std)
```

Input arguments

- Mean [numeric] - Mean of the beta distribution.
- Std [numeric] - Std dev of the beta distribution.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of the beta density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

gamma

Create function proportional to log of Chi-Squared distribution

Syntax

```
F = logdist.chisquare(Df)
```

Input arguments

- `Df` [integer] - Degrees of freedom of Chi-squared distribution.

Output arguments

- `F` [function_handle] - Function handle returning a value proportional to the log of the gamma density.

Description

See help on the logdisk package P196 for details on using the function handle `F`.

Example

---

gamma

Create function proportional to log of gamma distribution

Syntax

```
F = logdist.gamma(Mean,Std)
```

Input arguments

- Mean [numeric] - Mean of the gamma distribution.
- Std [numeric] - Std dev of the gamma distribution.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of the gamma density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

invgamma

Create function proportional to log of inv-gamma distribution

Syntax

```
F = logdist.invgamma(MEAN,STD)
```

Input arguments

- MEAN [numeric] - Mean of the inv-gamma distribution.
- STD [numeric] - Std dev of the inv-gamma distribution.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of the inv-gamma density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

lognormal

Create function proportional to log of log-normal distribution

Syntax

```
F = logdist.lognormal(Mean,Std)
```

Input arguments

- Mean [numeric] - Mean of the log-normal distribution.
- Std [numeric] - Std dev of the log-normal distribution.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of the log-normal density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

normal

Create function proportional to log of Normal distribution

Syntax

```
F = logdist.normal(Mean,Std,W)
```

Input arguments

- Mean [numeric] - Mean of the normal distribution.
- Std [numeric] - Std dev of the normal distribution.
- W [numeric] - Optional input containing mixture weights.

Multivariate cases are supported. Evaluating multiple vectors as an array of column vectors is supported.

If the mean and standard deviation are cell arrays then the distribution will be a mixture of normals. In this case the third argument is the vector of mixture weights.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of Normal density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

t

Create function proportional to log of Student T distribution

Syntax

```
F = logdist.t(Mean,Std,Df)
```

Input arguments

- Mean [numeric] - Mean of the normal distribution.
- Std [numeric] - Std dev of the normal distribution.
- Df [integer] - Number of degrees of freedom. If finite, the distribution is Student T; if omitted or Inf (default) the distribution is Normal.

Multivariate cases are supported. Evaluating multiple vectors as an array of column vectors is supported.

Output arguments

- F [function_handle] - Function handle returning a value proportional to the log of Normal or Student density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

---

uniform

Create function proportional to log of uniform distribution

Syntax

```
F = logdist.uniform(Lo,Hi)
```

Input arguments

- Lo [numeric] - Lower bound of the uniform distribution.
- Hi [numeric] - Upper bound of the uniform distribution.

Output arguments

- F [function_handle] - Handle to a function returning a value that is proportional to the log of the uniform density.

Description

See help on the logdisk package P196 for details on using the function handle F.

Example

12 Matrices with Named Rows and Columns (namedmat Objects)

Matrices with named rows and columns are returned as output arguments from several IRIS functions, such as model/acf P68, model/xsf P159, or model/fmse P99, to facilitate easy selection of submatrices by referrring to variable names in rows and columns.

Namedmat methods:

Constructor

- namedmat P205 - Create a new matrix with named rows and columns.

Manipulating named matrices

- select P207 - Select submatrix by referring to row names and column names.
- transpose P207 - Transpose each page of matrix with names rows and columns.

Getting row and column names

- rownames P206 - Names of rows in namedmat object.
- colnames P204 - Names of columns in namedmat object.

Sample characteristics

- [cutoff](namedmat/cutoff] -

All operators and functions available for standard Matlab matrices and arrays (i.e. double objects) are also available for namedmat objects.

Reference page for namedmat

---

colnames

Names of columns in namedmat object

Syntax

```
ColNames = colnames(X)
```

Input arguments

- X [namedmat] - A namedmat object (array with named rows and columns) returned as output argument from some model functions.

Output arguments

- ColNames [cellstr] - Names of columns in X.

Description

Example

---

namedmat

Create a new matrix with named rows and columns

Syntax

```
X = namedmat(X,RowNames,ColNames)
X = namedmat(X,Names)
```

Input arguments

- X [numeric] - Matrix or multidimensional array.
- RowNames [cellstr] - Names for individual rows of X.
- ColNames [cellstr] - Names for individual columns of X.
- Names [cellstr] - Names for both rows and columns of X.

Output arguments

- X [namedmat] - Matrix with named rows and columns.

Description

Namedmat objects are used by some of the IRIS functions to preserve the names of variables that relate to individual rows and columns, such as in

- acf, the autocovariance and autocorrelation functions,
- xsf, the power spectrum and spectral density functions,
- fmse, the forecast mean square error fuctions,
- etc.

You can use the function select P207 to extract submatrices by referring to a selection of names.

Namedmat matrices derives from the built-in double class of objects, and hence you can use any operators and functions on them that are available for double objects.

Example

---

rownames

Names of rows in namedmat object

Syntax

```
RowNames = rownames(X)
```

Input arguments

- X [namedmat] - A namedmat object (array with named rows and columns) returned as output argument from some model functions.

Output arguments

- RowNames [cellstr] - Names of rows in X.

Description

Example

---

select

Select submatrix by referring to row names and column names

Syntax

```
[XX,Pos] = select(X,RowSelect,ColSelect)
[XX,Pos] = select(X,Select)
```

Input arguments

- X [namedmat] - Matrix or array with named rows and columns.
- RowSelect [char | cellstr] - Selection of row names.
- ColSelect [char | cellstr] - Selection of column names.
- Select [char | cellstr] - Selection of names that will be applied to both rows and columns.

Output arguments

- XX [namedmat] - Submatrix with named rows and columns.
- Pos [cell] - Pos{1} is av ector of rows included in the submatrix XX, Pos{2} is a vector of columns included in the submatrixXX‘.

Description

Example

---

transpose

Transpose each page of matrix with names rows and columns

Syntax

```
X = transpose(X)
X = X.'
```

Input arguments

- X [namedmat] - Input matrix or array with named rows and columns.

Output arguments

- X [namedmat] - Transpose of the input matrix; if it is more than 2-dimensional, each page of the matrix is transposed.

Description

Example

Part III —
Multivariate Time Series Analysis

13 Vector Autoregressions (VAR Objects)

VAR objects can be constructed as plain VARs or simple panel VARs (with fixed effect), and estimated without or with prior dummy observations (quasi-bayesian VARs). VAR objects are reduced-form models; they are also the point of departure for identifying structural VARs (SVAR P250 objects).

VAR methods:

Constructor

- VAR P245 - Create new empty reduced-form VAR object.

Getting information about VAR objects

- addparam P213 - Add VAR parameters to a database (struct).
- comment P216 - Get or set user comments in an IRIS object.
- companion P216 - Matrices of first-order companion VAR.
- eig P217 - Eigenvalues of a VAR process.
- fprintf P224 - Write VAR model as formatted model code to text file.
- get P225 - Query VAR object properties.
- iscompatible P232 - True if two VAR objects can occur together on the LHS and RHS in an assignment.
- isexplosive P232 - True if any eigenvalue is outside unit circle.
- ispanel P233 - True for panel VAR objects.
- isstationary P234 - True if all eigenvalues are within unit circle.
- length P234 - Number of alternative parameterisations in VAR object.
- mean P235 - Mean of VAR process.
- nfitted P236 - Number of data points fitted in VAR estimation.
- rngcmp P239 - True if two VAR objects have been estimated using the same dates.
- sprintf P241 - Print VAR model as formatted model code.
- sspace P242 - Quasi-triangular state-space representation of VAR.
- userdata P244 - Get or set user data in an IRIS object.

Referencing VAR objects

- group P227 - Retrieve VAR object from panel VAR for specified group of data.
- subsasgn P243 - Subscripted assignment for VAR objects.
- subsref P244 - Subscripted reference for VAR objects.

Simulation, forecasting and filtering

- ferf P221 - Forecast error response function.
- filter P221 - Filter data using a VAR model.
- forecast P223 - Unconditional or conditional VAR forecasts.
- instrument P229 - Define forecast conditioning instruments in VAR models.
- resample P238 - Resample from a VAR object.
- simulate P240 - Simulate VAR model.

Manipulating VARs

- assign P214 - Manually assign system matrices to VAR object.
- alter P213 - Expand or reduce the number of alternative parameterisations within a VAR object.
- backward P215 - Backward VAR process.
- demean P217 - Remove constant and the effect of exogenous inputs from VAR object.
- horzcat P228 - Combine two compatible VAR objects in one object with multiple parameterisations.
- integrate P231 - Integrate VAR process and data associated with it.
- xasymptote P247 - Set or get asymptotic assumptions for exogenous inputs.

Stochastic properties

- acf P212 - Autocovariance and autocorrelation functions for VAR variables.
- fmse P222 - Forecast mean square error matrices.
- vma P247 - Matrices describing the VMA representation of a VAR process.
- xsf P248 - Power spectrum and spectral density functions for VAR variables.

Estimation, identification, and statistical tests

- estimate P218 - Estimate a reduced-form VAR or BVAR.
- infocrit P229 - Populate information criteria for a parameterised VAR.
- lrtest P235 - Likelihood ratio test for VAR models.
- portest P237 - Portmanteau test for autocorrelation in VAR residuals.
- schur P239 - Compute and store triangular representation of VAR.

Getting on-line help on VAR functions

```
help VAR
help VAR/function_name
```

Reference page for VAR

---

acf

Autocovariance and autocorrelation functions for VAR variables

Syntax

```
[C,R] = acf(V,...)
```

Input arguments

- `V` [VAR] - VAR object for which the ACF will be computed.

Output arguments

- `C` [namedmat | numeric] - Auto/cross-covariance matrices.
- `R` [namedmat | numeric] - Auto/cross-correlation matrices.

Options

- `'applyTo='` [logical | `@all`] - Logical index of variables to which the `'filter='` will be applied; `@all` means all variables.
- `'filter='` [char | empty] - Linear filter that is applied to variables specified by 'applyto'.
- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrices `C` and `R` as either `namedmat` P204 objects (i.e. matrices with named rows and columns) or plain numeric arrays.
- `'nFreq='` [numeric | `256`] - Number of equally spaced frequencies over which the `'filter='` is numerically integrated.
- `'order='` [numeric | `0`] - Order up to which ACF will be computed.
- `'progress='` [`true` | `false`] - Display progress bar in the command window.

Description

Example

---

addparam

Add VAR parameters to a database (struct)

Syntax

```
D = addparam(V,D)
```

Input arguments

- V [VAR] - VAR object whose parameter matrices will be added to database (struct) D.
- D [struct] - Database to which the model parameters will be added.

Output arguments

- 'D [struct] - Database with the VAR parameter matrices added.

Description

The newly created database entries are named A_ (transition matrix), K_ (constant terms), J_ (coefficient matrix in front of exogenous inputs), B_ (matrix of instantaneous whock effects), and Cov_ (covariance matrix of shocks). Be aware that all existing database entries in D named A_, K_, B_, or Omg_ will be overwritten.

Example

```
D = struct();
D = addparam(V,D);
```

---

alter

Expand or reduce the number of alternative parameterisations within a VAR object

Syntax

```
V = alter(V,N)
```

Input arguments

- V [VAR] - VAR object in which the number of paremeterisations will be changed.
- N [numeric] - New number of parameterisations.

Output arguments

- V [VAR] - VAR object with the new number of parameterisations.

Description

Example

---

assign

Manually assign system matrices to VAR object

Syntax

```
V = assign(V,A,K,J,Omg)
V = assign(V,A,K,J,Omg,Dates)
```

Input arguments

- V [VAR] - VAR object with variable names.
- A [numeric] - Transition matrices; see Description.
- K [numeric | empty] - Constant vector or matrix; if empty, the constant vector will be set to zeros, and will not be included in the number of free parameters.
- J [numeric | empty] - Coefficient matrix in front exogenous inputs; if empty the matrix will be set to zeros.
- Omg [numeric] - Covariance matrix of forecast errors (reduced-form residuals).
- Dates [numeric] - Vector of dates of (hypothetical) fitted observations; may be omitted.

Output arguments

- V [VAR] - VAR object with system matrices assigned.

Description

To assign matrices for a p-th order VAR, stack the transition matrices for individual lags horizontally,

```
A = [A1,...,Ap]
```

where A1 is the coefficient matrix on the first lag, and Ap is the coefficient matrix on the last, p-th, lag.

Example

---

backward

Backward VAR process

Syntax

```
B = backward(V)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- B [VAR] - VAR object with the VAR process reversed in time.

Description

Example

---

comment

Get or set user comments in an IRIS object

Syntax for getting user comments

```
Cmt = comment(Obj)
```

Syntax for assigning user comments

```
Obj = comment(Obj,Cmt)
```

Input arguments

- `Obj` [model | tseries | VAR | SVAR | FAVAR | sstate] - One of the IRIS objects.
- `Cmt` [char] - User comment that will be attached to the object.

Output arguments

- `Cmt` [char] - User comment that are currently attached to the object.

Description

Example

---

companion

Matrices of first-order companion VAR

Syntax

```
[A,B,K,J] = companion(V)
```

Input arguments

- `V` [VAR] - VAR object for which the companion matrices will be returned.

Output arguments

- A [numeric] - First-order companion transition matrix.
- B [numeric] - First-order companion coefficient matrix in front of reduced-form residuals.
- K [numeric] - First-order compnaion constant vector.

Description

Example

---

demean

Remove constant and the effect of exogenous inputs from VAR object

Syntax

```
V = demean(V)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- V [VAR] - VAR object with the constant vector, K, and the asymptotic assumptions for exogenous inputs, X0, reset to zero.

Description

Example

---

eig

Eigenvalues of a VAR process

Syntax

```
E = eig(V)
```

Input arguments

- V [VAR] - VAR object whose eigenvalues will be returned.

Output arguments

- E [numeric] - VAR eigenvalues.

Description

This function is equivalent to calling

```
e = get(v,'eig')
```

Example

---

estimate

Estimate a reduced-form VAR or BVAR

Syntax

```
[V,VData,Fitted] = estimate(V,Inp,Range,...)
```

Input arguments

- V [VAR] - Empty VAR object.
- Inp [struct] - Input database.
- Range [numeric] - Estimation range, including P pre-sample periods, where P is the order of the VAR.

Output arguments

- V [VAR] - Estimated reduced-form VAR object.
- VData [struct] - Output database with the endogenous variables and the estimated residuals.
- Fitted [numeric] - Dates for which fitted values have been calculated.

Options

- 'A=' [numeric | empty] - Restrictions on the individual values in the transition matrix, A.
- 'BVAR=' [numeric] - Prior dummy observations for estimating a BVAR; construct the dummy observations using the one of the BVAR functions.
- 'C=' [numeric | empty] - Restrictions on the individual values in the constant vector, C.
- 'J=' [numeric | empty] - Restrictions on the individual values in the coefficient matrix in front of exogenous inputs, J.
- 'diff=' [true | false] - Difference the series before estimating the VAR; integrate the series back afterwards.
- 'G=' [numeric | empty] - Restrictions on the individual values in the coefficient matrix in front of the co-integrating vector, G.
- 'cointeg=' [numeric | empty] - Co-integrating vectors (in rows) that will be imposed on the estimated VAR.
- 'comment=' [char | Inf] - Assign comment to the estimated VAR object; Inf means the existing comment will be preserved.
- 'constraints=' [char | cellstr] - General linear constraints on the VAR parameters.
- 'constant=' [true | false] - Include a constant vector in the VAR.
- 'covParam=' [true | false] - Calculate and store the covariance matrix of estimated parameters.
- 'eqtnByEqtn=' [true | false] - Estimate the VAR equation by equation.
- 'maxIter=' [numeric | 1] - Maximum number of iterations when generalised least squares algorithm is involved.
- 'mean=' [numeric | empty] - Impose a particular asymptotic mean on the VAR process.
- 'order=' [numeric | 1] - Order of the VAR.
- 'progress=' [true | false] - Display progress bar in the command window.

- 'schur=' [true | false] - Calculate triangular (Schur) representation of the estimated VAR straight away.
- 'stdize=' [true | false] - Adjust the prior dummy observations by the std dev of the observations.
- 'timeWeights=' [tseries | empty] - Time series of weights applied to individual periods in the estimation range.
- 'tolerance=' [numeric | 1e-5] - Convergence tolerance when generalised least squares algorithm is involved.
- 'warning=' [true | false] - Display warnings produced by this function.

Options for panel VAR

- 'fixedEff=' [true | false] - Include constant dummies for fixed effect in panel estimation; applies only if 'constant=' true.
- 'groupWeights=' [numeric | empty] - A 1-by-NGrp vector of weights applied to groups in panel estimation, where NGrp is the number of groups; the weights will be rescaled so as to sum up to 1.

Description

Estimating a panel VAR

Panel VAR objects are created by calling the function VAR P245 with two input arguments: the list of variables, and the list of group names. To estimate a panel VAR, the input data, Inp, must be organised a super-database with sub-databases for each group, and time series for each variables within each group:

```
d.Group1_Name.Var1_Name
d.Group1_Name.Var2_Name
...
d.Group2_Name.Var1_Name
d.Group2_Name.Var2_Name
...
```

Example

ferf

Forecast error response function

Syntax

```
[R,C] = ferf(V,NPer)
[R,C] = ferf(V,Range)
```

Input arguments

- V [VAR] - VAR object for which the forecast error response function will be computed.
- NPer [numeric] - Number of periods.
- Range [numeric] - Date range.

Output arguments

- Resp [tseries | struct] - Forecast error response functions.
- Cum [tseries | struct] - Cumulative forecast error response functions.

Options

- 'presample=' [true | false] - Include zeros for pre-sample initial conditions in the output data.
- 'select=' [cellstr | char | logical | numeric | Inf] - Selection of variable to whose forecast errors the responses will be simulated.

Description

Example

---

filter

Filter data using a VAR model

Syntax

```
[V,Outp] = filter(V,Inp,Range,...)
```

Input arguments

- `V` [VAR] - Input VAR object.
- `Inp` [struct] - Input database from which initial condition will be read.
- `Range` [numeric] - Forecast range.

Output arguments

- `V` [VAR] - Output VAR object.
- `Outp` [struct] - Output database with prediction and/or smoothed data.

Options

- `'cross='` [numeric | 1] - Multiply the off-diagonal elements of the covariance matrix (cross-covariances) by this factor; `'cross='` must be equal to or smaller than 1.
- `'deviation='` [`true` | `false`] - Both input and output data are deviations from the unconditional mean.
- `'meanOnly='` [`true` | `false`] - Return a plain database with mean forecasts only.
- `'omega='` [numeric | empty] - Modify the covariance matrix of residuals for this run of the filter.

Description

Example

---

fmse

Forecast mean square error matrices

Syntax

```
[F,X] = fmse(V,NPer)
[F,X] = fmse(V,Range)
```

Input arguments

- V [VAR] - VAR object for which the forecast MSE matrices will be computed.
- NPer [numeric] - Number of periods.
- Range [numeric] - Date range.

Output arguments

- F [namedmat | numeric] - Forecast MSE matrices.
- X [dbase | tseries] - Database or tseries with the std deviations of individual variables, i.e. the square roots of the corresponding diagonal elements of M.

Options

- 'matrixFmt=' ['namedmat' | 'plain'] - Return matrix F as either a namedmat P204 object (i.e. matrix with named rows and columns) or a plain numeric array.

---

forecast

Unconditional or conditional VAR forecasts

Syntax

```
Outp = forecast(V,Inp,Range,...)
Outp = forecast(V,Inp,Range,Cond,...)
```

Input arguments

- V [VAR] - VAR object.
- Inp [struct] - Input database from which initial condition will be read.
- Range [numeric] - Forecast range; must not refer to Inf.
- Cond [struct | tseries] - Conditioning database with the mean values of residuals, reduced-form conditions on endogenous variables, and conditioning instruments.

Output arguments

- Outp [struct] - Output database with forecasts of endogenous variables, residuals, and conditioning instruments.

Options

- 'cross=' [numeric | 1] - Multiply the off-diagonal elements of the covariance matrix (cross-covariances) by this factor; 'cross=' must be equal to or smaller than 1.
- 'dbOverlay=' [true | false] - Combine the output data with the input data; works only if the input data is a database.
- 'deviation=' [true | false] - Both input and output data are deviations from the unconditional mean.
- 'meanOnly=' [true | false] - Return a plain database with mean forecasts only.
- 'omega=' [numeric | empty] - Modify the covariance matrix of residuals for this forecast.

Description

Example

---

fprintf

Write VAR model as formatted model code to text file

Syntax

```
[C,D] = fprintf(V,File,...)
```

Input arguments

- V [VAR] - VAR object that will be printed to a model file.
- File [char | cellstr] - Filename, or filename format string, under which the model code will be saved.
- Output arguments
- C [cellstr] - Text string with the model code for each parameterisation.
- D [cell] - Parameter databases for each parameterisation; if 'hardParameters=' true, the database will be empty.

Options

See help on sprintf P241 for options available.

Description

For VAR objects with Na multiple alternative parameterisations, the filename File must be either a 1-by-Na cell array of string with a filename for each parameterisation, or a sprintf format string where a single occurence of '%g' will be replaced with the parameterisation number.

Example

---

get

Query VAR object properties

Syntax

```
Ans = get(V,Query)
[Ans,Ans,...] = get(V,Query,Query,...)
```

Input arguments

- V [VAR] - VAR object.
- Query [char] - Query to the VAR object.

Output arguments

- Ans [...] - Answer to the query.

Valid queries to VAR objects

VAR variables

- 'yList' – Returns [cellstr] the names of endogenous variables.
- 'eList' – Returns [cellstr] the names of residuals or shocks.
- 'iList' – Returns [cellstr] the names of conditioning (forecast) instruments.
- 'ny' – Returns [numeric] the number of variables.
- 'ne' – Returns [numeric] the number of residuals or shocks.
- 'ni' – Returns [numeric] the number of conditioning (forecast) instruments.

System matrices

- 'A#', 'A*', 'A$' – Returns [numeric] the transition matrix in one of the three possible forms; see Description.
- 'K', 'const' – Returns [numeric] the constant vector or matrix (the latter for panel VARs).
- 'J' – Returns [numeric] the coefficient matrix in front of exogenous inputs.
- 'Omg', 'Omega' – Returns [numeric] the covariance matrix of one-step-ahead forecast errors, i.e. reduced-form residuals. Note that this query returns the same matrix also for structural VAR (SVAR) objects.
- 'Sgm', 'Sigma' – Returns [numeric] the covariance matrix of the VAR parameter estimates; the matrix is non-empty only if the option 'covParam=' has been set to true at estimation time.
- 'G' – Returns [numeric] the coefficient matrix on cointegration terms.

Information criteria

- 'AIC' – Returns [numeric] Akaike information criterion.
- 'SBC' – Returns [numeric] Schwarz bayesian criterion.

Other queries

- `'cumLong'` – Returns [numeric] the matrix of long-run cumulative responses.
- `'nFree'` – Returns [numeric] the number of freely estimated (hyper-) parameters.
- `'order'`, `'p'` – Returns [numeric] the order of the VAR object.

Description

Transition matrix

There are three queries to request the VAR transition matrix: `'A#'`, `'A*'`, `'A$'`. They differ in how the higher-order transition matrices are arranged.

- `'A#'` returns `cat(3,I,-A1,...,-Ap)` where `I` is an identity matrix, and `A1, ... Ap` are the coefficient matrices on individual lags.
- `'A#'` returns `cat(3,A1,...,Ap)` where `A1, ... Ap` are the coefficient matrices on individual lags.
- `'A$'` returns `[A1,...,Ap]` where `A1, ... Ap` are the coefficient matrices on individual lags.

Example

---

group

Retrieve VAR object from panel VAR for specified group of data

Syntax

```
V = group(V,Grp)
```

Input arguments

- `V` [VAR] - Panel VAR object estimated on multiple groups of data.
- `Grp` [char] - Requested group name; must be one of the names specified when the panel VAR object was constructed using the function VAR P245.

Output arguments

- V [VAR] - VAR object for the K-th group of data.

Description

Example

Create and estimate a panel VAR for three variables, x, y, z, and three countries, US, EU, JA. Then, retrieve a plain VAR for an individual country.

```
v = VAR({'x','y','z'},{'US','EU','JA'});
v = estimate(v,d,range,'fixedEffect=',true);
vi_us = group(v,'US');
```

---

horzcat

Combine two compatible VAR objects in one object with multiple parameterisations

Syntax

```
V = [V1,V2,...]
```

Input arguments

- V1, V2 [VAR] - Compatible VAR objects that will be combined.

Output arguments

- V [VAR] - Output VAR object that combines the input VAR objects as multiple parameterisations.

Description

Example

---

infocrit

Populate information criteria for a parameterised VAR

Syntax

```
V = infocrit(V)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- V [VAR] - VAR object with the AIC and SBC information criteria re-calculated.

Description

In most cases, you don't have to run the function infocrit as it is called from within estimate immediately after a new parameterisation is created.

Example

---

instrument

Define forecast conditioning instruments in VAR models

Syntax to add forecast instruments

```
V = instrument(V,Def)
V = instrument(V,Name,Expr)
V = instrument(V,Name,Vec)
```

Syntax to remove all forecast instruments

```
V = instrument(V)
```

Input arguments

- V [VAR] - VAR object to which forecast instruments will be added.
- Def [char | cellstr] - Definition of the new forecast conditioning instrument.
- Name [char] - Name of the new forecast conditiong instrument.
- Expr [char] - Expression defining the new forecast conditiong instrument.
- Vec [numeric] - Vector of coeffients to combine the VAR variables to create the new forecast conditioning instrument.

Output arguments

- V [VAR] - VAR object with forecast instruments added or removed.

Description

Conditioning instruments allow you to compute forecasts conditional upon a linear combinationi of endogenous variables.

The definition strings must have the following form:

```
'name := expression'
```

where name is the name of the new conditioning instrument, and expression is an expression referring to existing VAR variable names and/or their lags.

Alternatively, you can separate the name and the expression into two input arguments. Or you can define the instrument by a vector of coefficients, either 1-by-N or 1-by-(N+1), where N is the number of variables in the VAR object V, and the last optional element is a constant term (set to zero if no value supplied).

The conditioning instruments must be a linear combination (possibly with a constant) of the existing endogenous variables and their lags up to p-1 where p is the order of the VAR. The names of the conditioning instruments must be unique (i.e. distinct from the names of endogenous variables, residuals, exogenous variables, and existing instruments).

Example

In the following example, we assume that the VAR object v has at least three endogenous variables named x, y, and z.

```
V = instrument(V,'i1 := x - x{-1}','i2: = (x + y + z)/3');
```

Note that the above line of code is equivalent to

```
V = instrument(V,'i1 := x - x{-1}');
V = instrument(V,'i2: = (x + y + z)/3');
```

The command defines two conditioning instruments named i1 and i2. The first instrument is the first difference of the variable x. The second instrument is the average of the three endogenous variables.

To impose conditions (tunes) on a forecast using these instruments, you run VAR/forecast P223 with the fourth input argument containing a time series for i1, i2, or both.

```
j = struct();
j.i1 = tseries(startdate:startdate+3,0);
j.i2 = tseries(startdate:startdate+3,[1;1.5;2]);

f = forecast(v,d,startdate:startdate+12,j);
```

■ integrate

Integrate VAR process and data associated with it

Syntax

```
V = integrate(V,...)
```

Input arguments

- V [VAR] - VAR object whose variables will be integrated by one order.

Output arguments

- V [VAR] - VAR object with the specified variables integrated by one order.

Options

- 'applyTo=' [logical | numeric | Inf] - Index of variables to integrate; Inf means all variables will be integrated.

Description

Example

■ iscompatible

True if two VAR objects can occur together on the LHS and RHS in an assignment

Syntax

```
Flag = iscompatible(V1,V2)
```

Input arguments

- `V1`, `V2` [model] - Two VAR objects that will be tested for compatibility.

Output arguments

- `Flag` [`true` | `false`] - True if `V1` and `V2` can occur in an assignment, `V1(...) = V2(...)`, or horizonatl concatenation, `[V1,V2]`.

Description

The function compares the names of all variables, shocks, and parameters, and the composition of the state-space vectors and matrices.

Example

■ isexplosive

True if any eigenvalue is outside unit circle

Syntax

```
Flag = isexplosive(V)
```

Input arguments

- V [VAR] - VAR object whose eigenvalues will be tested for explosiveness.

Output arguments

- Flag [true | false] - True if at least one eigenvalue is outside unit circle.

Options

- 'tolerance=' [numeric | getrealsmall()] - Tolerance for the eigenvalue test.

Description

Example

■ ispanel

True for panel VAR objects

Syntax

```
Flag = ispanel(X)
```

Input arguments

- X [VAR | SVAR] - VAR object.

Output arguments

- Flag [true | false] - True if the VAR object, X, is based on a panel of data.

Description

Example

---

isstationary

True if all eigenvalues are within unit circle

Syntax

```
Flag = isstationary(V)
```

Input arguments

- `V` [VAR] - VAR object whose eigenvalues will be tested for stationarity.

Output arguments

- `Flag` [`true` | `false`] - True if all eigenvalues are within unit circle.

Options

- `'tolerance='` [numeric | `getrealsmall()`] - Tolerance for the eigenvalue test.

---

length

Number of alternative parameterisations in VAR object

Syntax

```
N = length(V)
```

Input arguments

- `V` [VAR] - VAR object.

Output arguments

- `N` [numeric] - Number of alternative parameterisations.

Description

Example

---

lrtest

Likelihood ratio test for VAR models

Syntax

```
[Stat,Crit] = lrtest(V1,V2,Level)
```

Input arguments

- `V1` [VAR] - Unrestricted VAR model.
- `V2` [VAR] - Restricted VAR model.
- `Level` [numeric] - Significance level; if not specified, 5 percent significance is used, `Level=0.05`.

Output arguments

- `Stat` [numeric] - LR test stastic.
- `Crit` [numeric] - LR test critical value based on chi-square distribution.

Description

Example

---

mean

Mean of VAR process

Syntax

```
M = mean(V)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- M [numeric] - Asymptotic mean of the VAR variables.

Description

For plain VAR objects, the output argument X is a column vector where the k-th number is the asymptotic mean of the k-th variable, or NaN if the k-th variable is non-stationary (contains a unit root).

In panel VAR objects (with a total of Ng groups) and/or VAR objects with multiple alternative parameterisations (with a total of Na parameterisations), X is an Ny-by-Ng-by-Na matrix in which the column X(:,g,a) is the asyptotic mean of the VAR variables in the g-th group and the a-th parameterisation.

In VAR objects with exogenous inputs, the mean will be computed based on the asymptotic assumptions of exogenous inputs assigned by the function xasymptote P247.

Example

---

nfitted

Number of data points fitted in VAR estimation

Syntax

```
N = nfitted(V)
```

Input arguments

- V [VAR] - Estimated VAR object.

Output arguments

- N [numeric] - Number of data points (periods) fitted when estimating the VAR object.

Description

Example

---

portest

Portmanteau test for autocorrelation in VAR residuals

Syntax

```
[Stat,Crit] = portest(V,Data,H)
```

Input arguments

- V [VAR | swar] - Estimated VAR from which the tested residuals were obtained.
- Data [tseries] - VAR residuals, or VAR output data including residuals, to be tested for autocorrelation.
- H [numeric] - Test horizon; must be greater than the order of the tested VAR.

Output arguments

- Stat [numeric] - Portmanteau test statistic.
- Crit [numeric] - Portmanteau test critical value based on chi-square distribution.

Options

- 'level=' [numeric | 0.05] - Requested significance level for computing the criterion Crit.

Description

Example

---

resample

Resample from a VAR object

Syntax

```
Outp = resample(V,Inp,Range,NDraw,...)
Outp = resample(V,[ ],Range,NDraw,...)
```

Input arguments

- `V` [VAR] - VAR object to resample from.
- `Inp` [struct | tseries] - Input database or tseries used in bootstrap; not needed when `'method='` `'montecarlo'`.
- `Range` [numeric] - Range for which data will be returned.

Output arguments

- `Outp` [struct | tseries] - Resampled output database or tseries.

Options

- `'deviation='` [`true` | `false`] - Do not include the constant term in simulations.
- `'group='` [numeric | `NaN`] - Choose group whose parameters will be used in resampling; required in VAR objects with multiple groups when `'deviation='` `false`.
- `'method='` [`'bootstrap'` | `'montecarlo'` | function_handle] - Bootstrap from estimated residuals, resample from normal distribution, or use user-supplied sampler.
- `'progress='` [`true` | `false`] - Display progress bar in the command window.
- `'randomise='` [`true` | `false`] - Randomise or fix pre-sample initial condition.
- `'wild='` [`true` | `false`] - Use wild bootstrap instead of standard Efron bootstrap when `'method='` `'bootstrap'`.

Description

Example

---

rngcmp

True if two VAR objects have been estimated using the same dates

Syntax

```
Flag = rngcmp(V1,V2)
```

Input arguments

- V1, V2 [VAR] - Two estimated VAR objects.

Output arguments

- Flag [true | false] - True if the two VAR objects, V1 and V2, have been estimated using observations at the same dates.

Description

Example

---

schur

Compute and store triangular representation of VAR

Syntax

```
V = schur(V)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- V [VAR] - VAR object with the triangular representation matrices re-calculated.

Description

In most cases, you don't have to run the function schur as it is called from within estimate immediately after a new parameterisation is created.

Example

■ simulate

Simulate VAR model

Syntax

```
Outp = simulate(V,Inp,Range,...)
```

Input arguments

- V [VAR] - VAR object that will be simulated.
- Inp [tseries | struct] - Input data from which the initial condtions and residuals will be taken.
- Range [numeric] - Simulation range; must not refer to Inf.

Output arguments

- Outp [tseries] - Simulated output data.

Options

- 'contributions=' [true | false] - Decompose the simulated paths into the contributions of individual residuals, initial condition, the constant, and exogenous inputs; see Description.
- 'deviation=' [true | false] - Treat input and output data as deviations from unconditional mean.

- 'output=' ['auto' | 'dbase' | 'tseries'] - Format of output data.

Description

Backward simulation (backcast)

If the Range is a vector of decreasing dates, the simulation is performed backward. The VAR object is first converted to its backward representation using the function backward P215, and then the data are simulated from the latest date to the earliest date.

Simulation of contributions

With the option 'contributions=' true, the output database contains Ne+2 columns for each variable, where Ne is the number of residuals. The first Ne columns are the contributions of the individual shocks, the (Ne+1)-th column is the contribution of initial condition and the constant, and the last, (Ne+2)-th columns is the contribution of exogenous inputs.

Contribution simulations can be only run on VAR objects with one parameterization.

Example

---

sprintf

Print VAR model as formatted model code

Syntax

```
[C,D] = sprintf(V,...)
```

Input arguments

- V [VAR] - VAR object that will be printed as a formatted model code.
- Output arguments
- C [cellstr] - Text string with the model code for each parameterisation.
- D [cell] - Parameter database for each parameterisation; if 'hardParameters=' is true, the databases will be empty.

Options

- 'decimal=' [numeric | empty] - Precision (number of decimals) at which the coefficients will be written if 'hardParameters=' is true; if empty, the 'format=' options is used.
- 'declare=' [true | false] - Add declaration blocks and keywords for VAR variables, shocks, and equations.
- 'eNames=' [cellstr | char | empty] - Names that will be given to the VAR residuals; if empty, the names from the VAR object will be used.
- 'format=' [char | '%+.16g'] - Numeric format for parameter values; it will be used only if 'decimal=' is empty.
- 'hardParameters=' [true | false] - Print coefficients as hard numbers; otherwise, create parameter names and return a parameter database.
- 'yNames=' [cellstr | char | empty] - Names that will be given to the variables; if empty, the names from the VAR object will be used.
- 'tolerance=' [numeric | getrealsmall()] - Treat VAR coefficients smaller than 'tolerance=' in absolute value as zeros; zero coefficients will be dropped from the model code.

Description

Example

---

sspace

Quasi-triangular state-space representation of VAR

Syntax

```
[T,R,K,Z,H,D,Cov] = sspace(V,...)
```

Input arguments

- V [VAR] - VAR object.

Output arguments

- `T` [numeric] - Transition matrix.
- `R` [numeric] - Matrix of instantaneous effect of residuals (forecast errors).
- `K` [numeric] - Constant vector in transition equations.
- `Z` [numeric] - Matrix mapping transition variables to measurement variables.
- `H` [numeric] - Matrix at the shock vector in measurement equations (all zeros in VAR objects).
- `D` [numeric] - Constant vector in measurement equations (all zeros in VAR objects).
- `U` [numeric] - Transformation matrix for predetermined variables.
- `Cov` [numeric] - Covariance matrix of residuals (forecast errors).

Description

Example

■ subsasgn

Subscripted assignment for VAR objects

Syntax to assign parameterisations from other VAR object

```
V(inx) = W
```

Syntax to delete specified parameterisations

```
V(Inx) = []
```

Input arguments

- `V` [VAR] - VAR object.
- `inx` [numeric] - Index of parameterisations that will be assigned or deleted.
- `W` [VAR] - VAR object compatible with `V` whose parameterisations will be assigned (copied) into `V`.

Output arguments

- V [model] - VAR object with newly assigned or deleted parameterisations,

Description

Example

Expand the number of parameterisations in a VAR object that has initially just one parameterisation:

```
V(1:10) = V;
```

The parameterisation is simply copied ten times within the VAR object.

---

subsref

Subscripted reference for VAR objects

Syntax to retrieve VAR object with subset of parameterisations

```
V(Inx)
```

Input arguments

- V [VAR] - VAR object.
- Inx [numeric | logical] - Index of requested parameterisations.

Description

Example

---

userdata

Get or set user data in an IRIS object

Syntax for getting user data

```
X = userdata(Obj)
```

Syntax for assigning user data

```
OBJ = userdata(Obj,X)
```

Input arguments

- `Obj` [model | tseries | VAR | SVAR | FAVAR] - One of the IRIS objects with access to user data functions.
- `X` [...] - Any kind of data that will be attached to, and stored within, the object `OBJ`.

Output arguments

- `X` [...] - User data that are currently attached to the object.
- `Obj` [model | tseries | VAR | SVAR | FAVAR] - The object with its user data updated.

Description

Example

VAR

Create new empty reduced-form VAR object

Syntax for plain VAR and VARX

```
V = VAR(YNames)
V = VAR(YNames,'exogenous=',XNames)
```

Syntax for panel VAR and VARX

```
V = VAR(YNames,'groups=',GroupNames)
V = VAR(YNames,'exogenous=',XNames,'groups=',GroupNames)
```

Output arguments

- V [VAR] - New empty VAR object.
- YNames [cellstr | char | function_handle] - Names of endogenous variables.
- XNames [cellstr | char | function_handle] - Names of exogenous inputs.
- GroupNames [cellstr | char | function_handle] - Names of groups for panel VAR estimation.

Options

- 'exogenous=' [cellstr | empty] - Names of exogenous inputs; one of the names can be !ttrend, a linear time trend, which will be created automatically each time input data are required, and then included in the output database under the name ttrend.
- 'groups=' [cellstr | empty] - Names of groups for panel VAR estimation.

Description

This function creates a new empty VAR object. It is usually followed by an estimate P218 command to estimate the coefficient matrices in the VAR object using some data.

Example

To estimate a VAR, first create an empty VAR object specifying the variable names, and then run the VAR/estimate P218 function on it, e.g.

```
v = VAR({'x','y','z'});
[v,d] = estimate(v,d,range);
```

where the input database d ought to contain time series d.x, d.y, d.z.

---

vma

Matrices describing the VMA representation of a VAR process

Syntax

```
Phi = vma(V,N)
```

Input arguments

- V [VAR] - VAR object for which the VMA matrices will be computed.
- N [numeric] - Order up to which the VMA matrices will be computed.

Output arguments

- Phi [numeric] - VMA matrices.

Description

Example

---

xasymptote

Set or get asymptotic assumptions for exogenous inputs

Syntax

```
V = xasymptote(V,X0)
X = xasymptote(V)
```

Input arguments

- V [VAR] - VAR object.
- X0 [numeric] - A Nx-NGrp-by-NAlt vector or matrix of asymptotic assumptions for exogenous inputs, where Nx is the number of exogenous variables, NGrp is the number of groups in panel VARs, and NAlt is the number of alternative parameterizations.

Output arguments

- V [VAR] - VAR object.

Description

The asymptotic assumptions for exogenous inputs are used in the following contexts:

- to compute the asymptotic mean of the VAR process, mean P235;
- to set up initical conditions for resampling, resample P238, when they are not supplied in the input database.

If any of the three dimensions of the vector/matrix X0 is size 1, it will be automatically expanded to its appropriate size.

The asymptotic assumptions are reset to NaN each time the VAR object is estimated using the function estimate P218.

Example

---

xsf

Power spectrum and spectral density functions for VAR variables

Syntax

```
[S,D] = xsf(V,Freq,...)
```

Input arguments

- V [VAR] - VAR object.
- Freq [numeric] - Vector of Frequencies at which the XSFs will be evaluated.

Output arguments

- S [numeric] - Power spectrum matrices.
- D [numeric] - Spectral density matrices.

Options

- 'applyTo=' [cellstr | char | @all] - List of variables to which the 'filter=' will be applied; @all means all variables.
- 'filter=' [char | empty] - Linear filter that is applied to variables specified by 'applyto'.
- 'nFreq=' [numeric | 256] - Number of equally spaced frequencies over which the 'filter' is numerically integrated.
- 'progress=' [true | false] - Display progress bar in the command window.

Description

The output matrices, S and D, are N-by-N-by-K, where N is the number of VAR variables and K is the number of frequencies (i.e. the length of the vector freq).

The k-th page is the S matrix, i.e. S(:,:,k), is the cross-spectrum matrix for the VAR variables at the k-th frequency. Similarly, the k-th page in D, i.e. D(:,:,k), is the cross-density matrix.

Example

14 Structural Vector Autoregressions (SVAR Objects)

SVAR methods:

Constructor

- SVAR P255 - Convert reduced-form VAR to structural VAR.

SVAR objects can call any of the VAR P210 functions. In addition, the following functions are available for SVAR objects.

Getting information about SVAR objects

- get P252 - Query SVAR object properties.

Simulation

- srf P254 - Shock (impulse) response function.

Stochastic properties

- fevd P251 - Forecast error variance decomposition for SVAR variables.

Manipulating SVAR objects

- sort P253 - Sort SVAR parameterisations by squared distance of shock reponses to median.

See help on VAR P210 objects for other functions available.

Getting on-line help on SVAR functions

```
help SVAR
help SVAR/function_name
```

Getting on-line help on SVAR functions that are inherited from VARs

```
help VAR
help VAR/function_name
```

Reference page for SVAR

---

fevd

Forecast error variance decomposition for SVAR variables

Syntax

```
[X,Y,XX,YY] = fevd(V,NPer)
[X,Y,XX,YY] = fevd(V,Range)
```

Input arguments

- `V` [VAR] - Structural VAR model.
- `NPer` [numeric] - Number of periods.
- `Range` [numeric] - Date range.

Output arguments

- `X` [namedmat | numeric] - Forecast error variance decomposition into absolute contributions of residuals; absolute contributions sum up to the total variance.
- `Y` [namedmat | numeric] - Forecast error variance decomposition into relative contributions of residuals; relative contributions sum up to 1.
- `XX` [tseries] - Database with a tseries with absolute contributions in columns for each VAR variable.
- `YY` [tseries] - Database with a tseries with relative contributions in columns for each VAR variable.

Options

- `'matrixFmt='` [`'namedmat'` | `'plain'`] - Return matrices `X` and `Y` as be either `namedmat` P204 objects (i.e. matrices with named rows and columns) or plain numeric arrays.

Description

The output matrices X and Y are Ny-by-Ny-by-Nt-by-NAlt namedmat objects (matrices with named rows and columns), where Ny is the number of endogenous variables (and hence also structural residuals), Nt is the number of periods, and NAlt is the number of alternative parameterizations.

The output databases XX and YY contain Nt-by-Ny-by-NAlt tseries objects (one for each endogenous variable).

Example

---

get

Query SVAR object properties

Syntax

```
Ans = get(V,Query)
[Ans,Ans,...] = get(V,Query,Query,...)
```

Input arguments

- V [SVAR] - SVAR object.
- Query [char] - Query to the SVAR object.

Output arguments

- Ans [...] - Answer to the query.

Valid queries to SVAR objects

All queries to VAR objects, listed and described in VAR/get P225, can also be used in SVAR objects. In addition, the following queries are specific to SVAR objects:

- 'B' – Returns [numeric] matrix of instantaneous effects of shocks.
- 'std' – Returns [numeric] std deviation of structural shocks.

- 'method' – Returns [char] identification method used to convert reduced-form VAR to structural VAR.

Description

Example

■ sort

Sort SVAR parameterisations by squared distance of shock reponses to median

Syntax

```
[B,~,Inx,Crit] = sort(A,[],SortBy,...)
[B,Data,Inx,Crit] = sort(A,Data,SortBy,...)
```

Input arguments

- A [SVAR] - SVAR object with multiple parameterisations that will be sorted.
- Data [struct | empty] - SVAR database; if non-empty, the structural shocks will be re-ordered according to the SVAR parameterisations.
- SortBy [char] - Text string that will be evaluated to compute the criterion by which the parameterisations will be sorted; see Description for how to write SortBy.

Output arguments

- B [SVAR] - SVAR object with parameterisations sorted by the specified criterion.
- Data [tseries | struct | empty] - SVAR data with the structural shocks re-ordered to correspond to the order of parameterisations.
- Inx [numeric] - Vector of indices so that B = A(Inx).
- Crit [numeric] - The value of the criterion based on the string SortBy for each parameterisation.

Options

- 'progress=' [true | false] - Display progress bar in the command window.

Description

The individual parameterisations within the SVAR object A are sorted by the sum of squared distances of selected shock responses to the respective median reponses. Formally, the following criterion is evaluated for each parameterisation

$$\sum_{i \in I, j \in J, k \in K} [S_{i,j}(k) - M_{i,j}(k)]^2$$

where $S_{i,j}(k)$ denotes the response of the i-th variable to the j-th shock in period k, and $M_{i,j}(k)$ is the median responses. The sets of variables, shocks and periods, i.e. I, J, K, respectively, over which the summation runs are determined by the user in the SortBy string.

How do you select the shock responses that enter the criterion in SortBy? The input argument SortBy is a text string that refers to array S, whose element S(i,j,k) is the response of the i-th variable to the j-th shock in period k.

Note that when you pass in SVAR data and request them to be sorted the same way as the SVAR parameterisations (the second line in Syntax), the number of parameterisations in A must match the number of data sets in Data.

Example

Sort the parameterisations by squared distance to median of shock responses of all variables to the first shock in the first four periods. The parameterisation that is closest to the median responses

```
S2 = sort(S1,[],'S(:,1,1:4)')
```

---

srf

Shock (impulse) response function

Syntax

```
[Resp,Cum] = srf(V,NPer)
[Resp,Cum] = srf(V,Range)
```

Input arguments

- V [SVAR] - SVAR object for which the impulse response function will be computed.
- NPer [numeric] - Number of periods.
- Range [numeric] - Date range.

Output arguments

- Resp [tseries | struct] - Shock response functions.
- Cum [tseries | struct] - Cumulative shock response functions.

Options

- 'presample=' [true | false] - Include zeros for pre-sample initial conditions in the output data.
- 'select=' [cellstr | char | logical | numeric | Inf] - Selection of shocks to which the responses will be simulated.

Description

Example

SVAR

Convert reduced-form VAR to structural VAR

Syntax

```
[S,DATA,B,COUNT] = SVAR(V,DATA,...)
```

Input arguments

- V [VAR] - Reduced-form VAR object.
- DATA [struct | tseries] - Data associated with the input VAR object.

Output arguments

- S [VAR] - Structural VAR object.
- DATA [struct | tseries] - Data with transformed structural residuals.
- B [numeric] - Impact matrix of structural residuals.
- COUNT [numeric] - Number of draws actually performed (both successful and unsuccessful) when 'method'='draw'; otherwise COUNT=1.

Options

- 'maxIter=' [numeric | 0] - Maximum number of attempts when 'method'='draw'.
- 'method=' ['chol' | 'householder' | 'qr' | 'svd'] - Method that will be used to identify structural VAR and structural shocks.
- 'nDraw=' [numeric | 0] - Target number of successful draws when 'method'='draw'.
- 'reorder=' [numeric | empty] - Reorder VAR variables before identifying structural shocks, and bring the variables back in original order afterwards. Use the option 'backorderResiduals=' to control if also the structural shocks are to be brought back in original order.
- 'output=' ['auto' | 'dbase' | 'tseries'] - Format of output data.
- 'progress=' [true | false] - Display progress bar in the command window.
- 'rank=' [numeric | Inf] - Reduced rank of the covariance matrix of structural residuals when 'method=' 'svd'; Inf means full rank is preserved.
- 'backOrderResiduals=' [true | false] - Bring the identified structural shocks back in original order after identification; works with 'reorder='.
- 'std=' [numeric | 1] - Std deviation of structural residuals; the resulting structural covariance matrix will be re-scaled (divided) by this factor.
- 'test=' [char] - Works with 'method=draw' only; a string that will be evaluated for each random draw of the impact matrix B. The evaluation must result in true or false; only the matrices B that evaluate to true will be kept. See Description for more on how to write the option 'test='.

Description

Identification random Householder transformations

The structural impact matrices B are randomly generated using a Householder transformation algorithm. Each matrix is tested by evaluating the test string supplied by the user. If it evaluates to true the matrix is kept and one more SVAR parameterisation is created, if it is false the matrix is discarded.

The test string can refer to the following characteristics:

- S – the impulse (or shock) response function; the S(i,j,k) element is the response of the i-th variable to the j-th shock in period k.
- Y – the asymptotic cumulative response function; the Y(i,j) element is the asumptotic (long-run) cumulative response of the i-th variable to the j-th shock.

Example

15 Bayesian VAR Priors (BVAR Package)

The BVAR package is used to create basic types of prior dummy observations when estimating Bayesian VAR models. The dummy observations are passed in the VAR/estimate P218 function through the 'BVAR=' option.

Constructing dummy observations

- covmat P258 - Covariance matrix prior dummy observations for BVARs.
- litterman P259 - Litterman's prior dummy observations for BVARs.
- sumofcoeff P260 - Doan et al sum-of-coefficient prior dummy observations for BVARs.
- uncmean P260 - Unconditional-mean dummy (or Sims' initial dummy) observations for BVARs.
- user P261 - User-supplied prior dummy observations for BVARs.

Weights on prior dummy observations

The prior dummies produced by litterman P259, uncmean P260, sumofcoeff P260 can be weighted up or down using the input argument Mu. To give the weight a clear interpretation, use the option 'stdize=' true when estimating the VAR. In that case, setting Mu to sqrt(N) means the prior dummies are worth a total of extra N artifical observations; the weight can be related to the actual number of observations used in estimation.

Getting help on BVAR functions

```
help BVAR
help BVAR/function_name
```

---

covmat

Covariance matrix prior dummy observations for BVARs

Syntax

```
O = BVAR.covmat(C,Rep)
```

Input arguments

- C [numeric] - Prior covariance matrix of residuals; if C is a vector it will be converted to a diagonal matrix.
- Rep [numeric] - The number of times the dummy observations will be repeated.

Output arguments

- O [bvarobj] - BVAR object that can be passed into the VAR/estimate P218 function.

Description

Example

---

litterman

Litterman's prior dummy observations for BVARs

Syntax

```
O = BVAR.litterman(Rho,Mu,Lmb)
```

Input arguments

- Rho [numeric] - White-noise priors (Rho = 0) or random-walk priors (Rho = 1), or something in between.
- Mu [numeric] - Weight on dummy observations.
- Lmb [numeric] - Exponential increase in weight depending on the lag; Lmb = 0 means all lags are weighted equally.

Output arguments

- O [bvarobj] - BVAR object that can be passed into the VAR/estimate P218 function.

Description

See the section explaining the weights on prior dummies P258, i.e. the input argument Mu.

Example

---

sumofcoeff

Doan et al sum-of-coefficient prior dummy observations for BVARs

Syntax

```
O = BVAR.sumofcoeff(Mu)
```

Input arguments

- Mu [numeric] - Weight on the dummy observations.

Output arguments

- O [bvarobj] - BVAR object that can be passed into the VAR/estimate P218 function.

Description

See the section explaining the weights on prior dummies P258, i.e. the input argument Mu.

Example

---

uncmean

Unconditional-mean dummy (or Sims' initial dummy) observations for BVARs

Syntax

```
O = BVAR.uncmean(YBar,Mu)
```

Input arguments

- `YBar` [numeric] - Vector of unconditional means imposed as priors.
- `Mu` [numeric] - Weight on the dummy observations.

Output arguments

- `X` [numeric] - Array with prior dummy observations that can be used in the `'BVAR='` option of the `VAR/estimate` P218 function.
- `O` [bvarobj] - BVAR object that can be passed into the `VAR/estimate` P218 function.

Description

See the section explaining the weights on prior dummies P258, i.e. the input argument `Mu`.

Example

---

user

User-supplied prior dummy observations for BVARs

Syntax

```
O = BVAR.user(Y0,K0,Y1,G1)
```

Input arguments

- `Y0` [numeric] - Column-wise prior dummy observations on the LHS.
- `K0` [numeric] - Column-wise prior dummy observations on the RHS constant.

- Y1 [numeric] - Column-wise prior dummy observations on the RHS lagged variables.
- G1 [numeric] - Column-wise prior dummy observations on the RHS coefficients on the co-integrating vector.

Output arguments

- 0 [bvarobj] - BVAR object that can be passed into the VAR/estimate P218 function.

Description

Example

16 Factor-Augmented Vector Autoregressions (FAVAR Objects)

Constructor

- FAVAR P265 - Create new empty FAVAR object.

Getting information about FAVAR objects

- comment P263 - Get or set user comments in an IRIS object.
- get P268 - Query model object properties.
- isempty P270 - True if VAR based object is empty.
- userdata P271 - Get or set user data in an IRIS object.
- VAR P271 - Return a VAR object describing the factor dynamics.

Estimation

- estimate P264 - Estimate FAVAR using static principal components.

Filtering and forecasting

- filter P266 - Re-estimate the factors by Kalman filtering the data taking FAVAR coefficients as given.
- forecast P268 - Forecast FAVAR factors and observables.

Getting on-line help on FAVAR functions

```
help FAVAR
help FAVAR/function_name
```

Reference page for FAVAR

■ comment

Get or set user comments in an IRIS object

Syntax for getting user comments

```
Cmt = comment(Obj)
```

Syntax for assigning user comments

```
Obj = comment(Obj,Cmt)
```

Input arguments

- `Obj` [model | tseries | VAR | SVAR | FAVAR | sstate] - One of the IRIS objects.
- `Cmt` [char] - User comment that will be attached to the object.

Output arguments

- `Cmt` [char] - User comment that are currently attached to the object.

Description

Example

■ estimate

Estimate FAVAR using static principal components

Syntax

```
[A,D,CC,F,U,E,CTF] = estimate(A,D,Range,[R,Q],...)
```

Input arguments

- `A` [FAVAR] - Empty FAVAR object.
- `D` [struct] - Input database.
- `Range` [numeric] - Estimation range.
- `R` [numeric] - Selection criterion for the number of factors: Minimum requested proportion of input data volatility explained by the factors.
- `Q` [numeric] - Selection criterion for the number of factors: Maximum number of factors.

Output arguments

- A [FAVAR] - Estimated FAVAR object.
- D [struct] - Output database.
- CC [tseries] - Estimates of common components in the FAVAR observables.
- F [tseries] - Estimates of factors.
- U [struct | tseries] - Idiosyncratic residuals.
- E [tseries] - Factor VAR residuals.
- CTF [tseries] - Contributions of individual input series to the estimated factors.

Options

- 'cross=' [true | false | numeric] - Keep off-diagonal elements in the covariance matrix of idiosyncratic residuals; if false all cross-covariances are reset to zero; if a number between zero and one, all cross-covariances are multiplied by that number.
- 'order=' [numeric | 1] - Order of the VAR for factors.
- 'output=' ['auto' | 'dbase' | 'tseries'] - Format of output data.
- 'rank=' [numeric | Inf] - Restriction on the rank of the factor VAR residuals.

Description

Example

FAVAR

Create new empty FAVAR object

Syntax

```
F = FAVAR(YNames)
```

Input arguments

- YNames [cellstr | char] - Names of observed variables in the FAVAR model.

Output arguments

- F [FAVAR] - New FAVAR object.

Description

This function creates a new empty FAVAR object. It is usually followed by the estimate P264 function to estimate the FAVAR parameters on data.

Example

To estimate a FAVAR, you first need to create an empty VAR object, and then run the FAVAR P264 function on it, e.g.

```
list = {'DLCPI','DLGDP','R'};
f = FAVAR(list);
f = estimate(f,d,range);
```

---

filter

Re-estimate the factors by Kalman filtering the data taking FAVAR coefficients as given

Syntax

```
[A,D,CC,F,U,E] = filter(A,D,Range,...)
```

Input arguments

- A [FAVAR] - Estimated FAVAR object.
- D [struct | tseries] - Input database or tseries object with the FAVAR observables.
- Range [numeric] - Filter date range.

Output arguments

- A [FAVAR] - FAVAR object.
- D [struct] - Output database or tseries object with the FAVAR observables.
- CC [struct | tseries] - Re-estimated common components in the observables.
- F [tseries] - Re-estimated common factors.
- U [tseries] - Re-estimated idiosyncratic residuals.
- E [tseries] - Re-estimated structural residuals.

Options

- 'cross=' [true | false | numeric] - Run the filter with the off-diagonal elements in the covariance matrix of idiosyncratic residuals; if false all cross-covariances are reset to zero; if a number between zero and one, all cross-covariances are multiplied by that number.
- 'invFunc=' ['auto' | function_handle] - Inversion method for the FMSE matrices.
- 'meanOnly=' [true | false] - Return only mean data, i.e. point estimates.
- 'persist=' [true | false] - If filter or forecast is used with 'persist=' set to true for the first time, the forecast MSE matrices and their inverses will be stored; subsequent calls of the filter or forecast functions will re-use these matrices until filter or forecast is called.
- 'output=' ['auto' | 'dbase' | 'tseries'] - Format of output data.
- 'tolerance=' [numeric | 0] - Numerical tolerance under which two FMSE matrices computed in two consecutive periods will be treated as equal and their inversions will be re-used, not re-computed.

Description

It is the user's responsibility to make sure that filter and forecast called with 'persist=' set to true are valid, i.e. that the previously computed FMSE matrices can be really re-used in the current run.

Example

forecast

Forecast FAVAR factors and observables

Syntax

```
[D,CC,F,U,E] = forecast(A,D,RANGE,J,...)
```

Input arguments

- A [FAVAR] - FAVAR object.
- D [struct | tseries] - Input data with initial condition for the FAVAR factors.
- RANGE [numeric] - Forecast range.
- J [struct | tseries] - Conditioning data with hard tunes on the FAVAR observables.

Output arguments

- D [struct] - Output database or tseries object with the FAVAR observables.
- CC [struct | tseries] - Projection of common components in the observables.
- F [tseries] - Projection of common factors.
- U [tseries] - Conditional idiosyncratic residuals.
- E [tseries] - Conditional structural residuals.

Options

See help on FAVAR/filter P266 for options available.

Description

Example

---

get

Query model object properties

Syntax

```
Ans = get(A,Query)
[Ans,Ans,...] = get(A,Query,Query,...)
```

Input arguments

- A [FAVAR] - FAVAR object.
- Query [char] - Query to the FAVAR object.

Output arguments

- Ans [...] - Answer to the query.

Valid queries to FAVAR objects

System matrices

- 'A*' Returns [numeric] the transition matrix of the underlying VAR system on factors.
- 'B' Returns [numeric] tne matrix mapping the impact of structural residuals on the factors in the underlying VAR.
- 'C' Returns [numeric] the matrix mapping the factors into the observables.
- 'Omega' Returns [numeric] the reduced-form covariance matrix of the residuals in the underlying VAR.
- 'Sigma' Returns [numeric] the covariance matrix of idiosyncratic shocks.

Underlying VAR

- 'VAR' Returns [VAR] a VAR object describing the factor dynamics.

Eigenvalues and singular values

- 'eig' Returns [numeric] the vector of eigenvalues of the underlying VAR.
- 'sing' Returns [numeric] the vector of singular values from the principal component estimation step.

Observables and factors

- 'mean' Returns [numeric] the estimated mean of the observables used to standardise the input data.
- 'std' Returns [numeric] the estimated std deviations of the observables used to standardise the input data.
- 'ny' Returns [numeric] the number of observables.
- 'nx' Returns [numeric] the number of factors.
- 'yList' Returns [cellstr] the list of the names of observables.

Description

Example

---

isempty

True if VAR based object is empty

Syntax

```
Flag = isempty(X)
```

Input arguments

- X [VAR | SVAR | FAVAR] - VAR based object.

Output argument

- Flag [true | false] - True if the VAR based object, X, is empty.

Description

Example

---

userdata

Get or set user data in an IRIS object

Syntax for getting user data

```
X = userdata(Obj)
```

Syntax for assigning user data

```
OBJ = userdata(Obj,X)
```

Input arguments

- `Obj` [model | tseries | VAR | SVAR | FAVAR] - One of the IRIS objects with access to user data functions.
- `X` [...] - Any kind of data that will be attached to, and stored within, the object `OBJ`.

Output arguments

- `X` [...] - User data that are currently attached to the object.
- `Obj` [model | tseries | VAR | SVAR | FAVAR] - The object with its user data updated.

Description

Example

VAR

Return a VAR object describing the factor dynamics

Syntax

```
v = VAR(a)
```

Input arguments

a [FAVAR] - FAVAR object.

Output arguments

v [VAR] - VAR object describing the dynamic system of the FAVAR factors.

Description

Example

Part IV —
Time Series and Database Management

17 Dates and Date Ranges

Creating IRIS serial date numbers

- bb P275 - IRIS serial date number for bimonthly date.
- bbtoday P276 - IRIS serial date number for current bi-month.
- hh P295 - IRIS serial date number for half-yearly date.
- hhtoday P295 - IRIS serial date number for current half-year.
- mm P296 - IRIS serial date number for monthly date.
- mmtoday P297 - IRIS serial date number for current month.
- qq P297 - IRIS serial date number for quarterly date.
- qqtoday P298 - IRIS serial date number for current quarter.
- ww P302 - IRIS serial date number for weekly date.
- wwtoday P304 - IRIS serial date number for current week.
- yy P304 - IRIS serial date number for yearly date.
- yytoday P305 - IRIS serial date number for current year.

Computing special dates (daily dates only)

- datbom P285 - Beginning of month for the specified daily date.
- datboq P285 - Beginning of quarter for the specified daily date.
- datboy P286 - Beginning of year for the specified daily date.
- dateom P288 - End of month for the specified daily date.
- dateoq P289 - End of quarter for the specified daily date.
- dateoy P289 - End of year for the specified daily date.

Creating date ranges

- datrange P290 - Numerically safe way to create a date range.
- dat2ttrend P283 - Construct linear time trend from date range.
- datxtick P291 - Change ticks, labels and/or date frequency on x-axis in existing tseries graphs.

Converting dates

- clp2dat P276 - Convert text in system clipboard to dates.
- dat2char P277 - Convert dates to character array.
- dat2charlist P278 - Convert dates to a comma-separated list.
- dat2clp P279 - Convert dates to text and paste to system clipboard.
- dat2dec P279 - Convert dates to decimal grid.
- dat2str P280 - Convert IRIS dates to cell array of strings.
- dat2ypf P284 - Convert IRIS serial date number to year, period and frequency.

- dec2dat P294 - Convert decimal representation of date to IRIS serial date number.
- str2dat P299 - Convert strings to IRIS serial date numbers.
- textinp2dat P300 - Convert text input to IRIS serial date numbers.

Date comparison

- datcmp P286 - Compare two IRIS serial date numbers.
- datdiff P287 - Number of periods between two dates with check for date frequency.
- rngcmp P298 - Compare two IRIS date ranges.

Daily and weekly dates

- daysinyear P292 - Number of days in year.
- dd P293 - Matlab serial date numbers that can be used to construct daily tseries objects.
- ddtoday P294 - Matlab serial date number for today's date.
- ww2day P303 - Convert weekly IRIS serial date number to Matlab serial date number.
- weeksinyear P301 - Number of weeks in year.

Getting on-line help on date functions

```
help dates
help dates/function_name
```

---

bb

IRIS serial date number for bimonthly date

Syntax

```
Dat = bb(Y)
Dat = bb(Y,B)
```

Input arguments

- `Y` [numeric] - Years.
- `B` [numeric] - Bimonth; if omitted, first bimonth (January-February) is assumed.

Output arguments

- Dat [numeric] - IRIS serial date numbers representing the bimonthly date.

Description

Example

---

bbtoday

IRIS serial date number for current bi-month

Syntax

```
Dat = bbtoday()
```

Output arguments

- Dat [numeric] - IRIS serial date number for current bi-month.

Description

Example

---

clp2dat

Convert text in system clipboard to dates

Syntax

```
D = clp2dat(...)
```

Output arguments

- D [numeric] - IRIS serial date numbers based on the current content of the system clipboard converted by the str2dat P299 function.

Options

See help on str2dat P299 for options available.

Description

Example

---

dat2char

Convert dates to character array

Syntax

```
C = dat2char(Dat,...)
```

Input arguments

- Dat [numeric] - IRIS serial date numbers that will be converted to character array.

Output arguments

- C [char] - Character array representing the input dates; each line of the array represents one date from D.

Options

See help on dat2str P280 for options available.

Description

Example

We create a quarterly date using the function qq; this function returns an IRIS serial date number. We then use dat2char to print a humna-readable text representation of that date.

```
d = qq(2015,3)
d =
   8.0620e+03
dat2char(d)
ans =
2015Q3
```

---

dat2charlist

Convert dates to a comma-separated list

Syntax

```
C = dat2charlist(D,...)
```

Input arguments

- D [numeric] - IRIS serial date numbers that will be converted to a comma-separated list.

Output arguments

- C [char] - Text string with a comma-separated list of dates.

Options

See help on dat2str P280 for options available.

Description

Example

---

dat2clp

Convert dates to text and paste to system clipboard

Syntax

```
C = dat2clp(D,...)
```

Input arguments

- `D` [numeric] - IRIS serial date numbers that will be converted to character array and pasted to the system clipboard.

Output arguments

- `C` [char] - Character array representing the input dates pasted to the system clipboard; each line of the array represents one date from `D`.

Options

See help on `dat2str` P280 for options available.

Description

Example

---

dat2dec

Convert dates to decimal grid

Syntax

```
Dec = dat2dec(Dat)
Dec = dat2dec(Dat,Pos)
```

Input arguments

- `Dat` [numeric] - IRIS serial date number.
- `Pos` [`'start'` | `'centre'` | `'end'`] - Point within the period that will represent the date; if omitted, `Pos` is set to `'start'`.

Output arguments

- `Dec` [numeric] - Decimal grid representing the input dates, computed as `Year + (Per-1)/Freq`.

Description

Example

---

dat2str

Convert IRIS dates to cell array of strings

Syntax

```
S = dat2str(Dat,...)
```

Input arguments

- `Dat` [numeric] - IRIS serial date number(s).

Output arguments

- `S` [cellstr] - Cellstr with strings representing the input dates.

Options

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months.
- 'standinMonth=' [numeric | 'last' | 1] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.
- 'wwDay=' ['Mon' | 'Tue' | 'Wed' | 'Thu' | 'Fri' | 'Sat' | 'Sun'] - Day of week that will represent weeks.

Description

There are two types of date strings in IRIS: year-period strings and calendar date strings. The year-period strings can be printed for dates with yearly, half-yearly, quarterly, bimonthly, monthly, weekly, and indeterminate frequencies. The calendar date strings can be printed for dates with weekly and daily frequencies. Date formats for calendar date strings must start with a dollar sign, $.

Year-period date strings

Regular date formats can include any combination of the following fields:

- 'Y' - Year.
- 'YYYY' - Four-digit year.
- 'YY' - Two-digit year.
- 'P' - Period within the year (half-year, quarter, bi-month, month, week).
- 'PP' - Two-digit period within the year.
- 'R' - Upper-case roman numeral for the period within the year.
- 'r' - Lower-case roman numeral for the period within the year.
- 'M' - Month numeral.
- 'MM' - Two-digit month numeral.

- 'MMMM', 'Mmmm', 'mmmm' - Case-sensitive name of month.
- 'MMM', 'Mmm', 'mmm' - Case-sensitive three-letter abbreviation of month.
- 'Q' - Upper-case roman numeral for the month or stand-in month.
- 'q' - Lower-case roman numeral for the month or stand-in month.
- 'F' - Upper-case letter representing the date frequency.
- 'f' - Lower-case letter representing the date frequency.
- 'EE' - Two-digit end-of-month day; stand-in month used for non-monthly dates.
- 'E' - End-of-month day; stand-in month used for non-monthly dates.
- 'WW' - Two-digit end-of-month workday; stand-in month used for non-monthly dates.
- 'W' - End-of-month workday; stand-in month used for non-monthly dates.

Calendar date strings

Calendar date formats must start with a dollar sign, $, and can include any combination of the following fields:

- 'Y' - Year.
- 'YYYY' - Four-digit year.
- 'YY' - Two-digit year.
- 'DD' - Two-digit day numeral; daily and weekly dates only.
- 'D' - Day numeral; daily and weekly dates only.
- 'M' - Month numeral.
- 'MM' - Two-digit month numeral.
- 'MMMM', 'Mmmm', 'mmmm' - Case-sensitive name of month.
- 'MMM', 'Mmm', 'mmm' - Case-sensitive three-letter abbreviation of month.
- 'Q' - Upper-case roman numeral for the month.
- 'q' - Lower-case roman numeral for the month.
- 'DD' - Two-digit day numeral.
- 'D' - Day numeral.
- 'Aaa', 'AAA' - Three-letter English name of the day of week ('Mon', ..., 'Sun').

Escaping control letters

To get the format letters printed literally in the date string, use a percent sign as an escape character: `'%Y'`, `'%P'`, `'%F'`, `'%f'`, `'%M'`, `'%m'`, `'%R'`, `'%r'`, `'%Q'`, `'%q'`, `'%D'`, `'%E'`, `'%D'`.

Example

---

dat2ttrend

Construct linear time trend from date range

Syntax

```
[TTrend,BaseDate] = dat2ttrend(Range)
[TTrend,BaseDate] = dat2ttrend(Range,BaseYear)
[TTrend,BaseDate] = dat2ttrend(Range,Obj)
```

Input arguments

- `Range` [numeric] - Date range from which an integer linear time trend will be constructed.
- `BaseYear` [model | VAR] - Base year that will be used to construct the time trend.
- `Obj` [model | VAR] - Model or VAR object whose base year will be used to construct the time trend; if both `BaseYear` and `Obj` are omitted, the base year from `irisget('baseYear')` will be used.

Output arguments

- `TTrend` [numeric] - Integer linear time trend, unique to the input date range `Range` and the base year.
- `BaseDate` [numeric] - Base date used to normalize the input date range; see Description.

Description

For regular date frequencies, the time trend is constructed the following way. First, a base date is created first period in the base year of a given frequency. For instance, for a quarterly input range,

`BaseDate = qq(baseYear,1)`, for a monthly input range, `BaseDate == mm(baseYear,1)`, etc. Then, the output trend is an integer vector normalized to the base date,

```
TTrend = floor(Range - BaseDate);
```

For indeterminate date frequencies, `BaseDate = 0`, and the output time trend is simply the input date range.

Example

---

dat2ypf

Convert IRIS serial date number to year, period and frequency

Syntax

```
[Y,P,F] = dat2ypf(Dat)
```

Input arguments

- `Dat` [numeric] - IRIS serial date numbers.

Output arguments

- `Y` [numeric] - Years.
- `P` [numeric] - Periods within year.
- `F` [numeric] - Date frequencies.

Description

Example

---

datbom

Beginning of month for the specified daily date

Syntax

```
Bom = datebom(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- `Bom` [numeric] - Daily serial date number for the first day of the same month as `D`.

Description

Example

---

datboq

Beginning of quarter for the specified daily date

Syntax

```
Boq = datboq(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- `Boq` [numeric] - Daily serial date number for the first day of the same quarter as `D`.

Description

Example

---

datboy

Beginning of year for the specified daily date

Syntax

```
Boy = dateboy(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- `Boy` [numeric] - Daily serial date number for the first day of the same year as `D`.

Description

Example

---

datcmp

Compare two IRIS serial date numbers

Syntax

```
Flag = datcmp(Dat1,Dat2)
```

Input arguments

- `Dat1`, `Dat2` [numeric] - IRIS serial date numbers or vectors.

Output arguments

- Flag [true | false] - True for numbers that represent the same date.

Description

The two date vectors must either be the same lengths, or one of them must be scalar.

Use this function instead of the plain comparison operator, ==, to compare dates. The plain comparision can sometimes give false results because of round-off errors.

Example

```
d1 = qq(2010,1);
d2 = qq(2009,1):qq(2010,4);
datcmp(d1,d2)
ans =
    0     0     0     0     1     0     0     0
```

---

datdiff

Number of periods between two dates with check for date frequency

Syntax

```
D = datdiff(D1,D2)
```

Input arguments

- D1, D2 [numeric] - IRIS dates of vectors of IRIS dates.

Output arguments

- D [numeric] - Number of periods between D1 and D2, positive for D1 greater than D2, negative for D1 smaller than D2, or NaN for dates of different frequencies.

Description

Example

```
d1 = mm(2010,12);
d2 = mm(2011,12);

datdiff(d1,d2)
ans =
   -12

datdiff(d2,d1)
ans =
    12

d3 = yy(2011);
datdiff(d1,d3)
ans =
   NaN
```

---

dateom

End of month for the specified daily date

Syntax

```
Eom = dateom(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- `Eom` [numeric] - Daily serial date number for the last day of the same month as `D`.

Description

Example

---

dateoq

End of quarter for the specified daily date

Syntax

```
Eoq = dateoq(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- `Eoq` [numeric] - Daily serial date number for the last day of the same quarter as `D`.

Description

Example

---

dateoy

End of year for the specified daily date

Syntax

```
Eoy = dateoy(D)
```

Input arguments

- `D` [numeric] - Daily serial date number.

Output arguments

- Eoy [numeric] - Daily serial date number for the last day of the same year as D.

Description

Example

■ datrange

Numerically safe way to create a date range

Syntax

```
Rng = datrange(Start,End)
Rng = datrange(Start,End,Step)
```

Input arguments

- Start [numeric] - Start date of the range.
- End [numeric] - End date of the range.
- Step [numeric] - Step size in the number of base periods; if omitted, Step = 1.

Output arguments

- Rng [numeric] - Date vector Start : Step : End.

Description

Most of the time, using a colon operator to create a date range works fine,

```
Start : Step : End
```

Under some (rather rare) circumstances, the colon operator may give incorrect results caused by rounding error difficulties since IRIS serial date numbers are non-integer values. In that case, the function datrange provides a safe workaround:

```
datrange(Start,End,Step)
```

is equivalent (but numerically safer) to

```
Start : Step : End
```

Example

The date ranges created in this example are identical, and no numerical inaccuracies exist:

```
r1 = qq(2000,1) : qq(2010,4);
r2 = datrange(qq(2000,1),qq(2010,4));
format long
r1 - r2
```

---

datxtick

Change ticks, labels and/or date frequency on x-axis in existing tseries graphs

Syntax

```
datxtick(Range,...)
datxtick(Ax,Range,...)
```

Input arguments

- `Ax` [numeric] - Handle to the axes object where the changes will be made; if not specified, the current axes object, `gca()`, is changed.
- `Range` [numeric] - New date range to which the x-axis will be changed.

Options

- `'datePosition='` [`'start'` | `'centre'` | `'end'`] - Where within each given period the date tick will be placed (at the beginning of the period, in the middle of the period, or at the end of the period).

- 'dateTicks=' [numeric | Inf] - Individual date ticks; if Inf, the ticks will be determined automatically using the standard Matlab algorithm.

See dat2str P280 for date formatting options available.

Description

Example

Create a graph plotting a quarterly series, and then change the ticks and labels on the x-axis to monthly:

```
x = tseries(qq(2010,1):qq(2011,4),@rand);
plot(x);
datxtick(mm(2010,1):mm(2011,12),'dateFormat=','Mmm YYYY');
```

---

daysinyear

Number of days in year

Syntax

```
N = daysinyear(Year)
```

Input arguments

- Year [numeric] - Year.

Output arguments

- N [numeric] - Number of days in Year.

Description

N is 365 for non-leap years, and 366 for leap years. Leap years are either years divisible by 4 but not 100, or years divisible by 400.

Example

```
daysinyear([2000,2200])
ans =
   366   365
```

---

dd

Matlab serial date numbers that can be used to construct daily tseries objects

Syntax

```
Dat = dd(Year,Month,Day)
Dat = dd(Year,Month,'end')
Dat = dd(Year,Month)
Dat = dd(Year)
```

Output arguments

- `Dat` [numeric] - IRIS serial date numbers.

Input arguments

- `Year` [numeric] - Year.
- `Month` [numeric | char | cellstr] - Calendar month in year; if missing, `Month` is `1` by default; `Month` can be also specified as a three-letter English abbreviation: `'Jan'`, `'Feb'`, ... `'Dec'`.
- `Day` [numeric] - Calendar day in month; if missing, `Day` is `1` by default; `'end'` means the end day of the respective month.

Description

Example

```
>> d = dd(2010,12,3)
d =
      734475
```

```
>> dat2str(d)
ans =
    '2010-Dec-03'
```

---

ddtoday

Matlab serial date number for today's date

Syntax

```
Dat = ddtoday()
```

Output arguments

- `Dat` [numeric] - Matlab serial date number for today's date.

Description

Example

---

dec2dat

Convert decimal representation of date to IRIS serial date number

Syntax

```
Dat = dec2dat(Dec,Freq)
```

Input arguments

- `Dec` [numeric] - Decimal numbers representing dates.
- `Freq` [freq] - Date frequency.

Output arguments

- Dat [numeric] - IRIS serial date numbers corresponding to the decimal representations Dec.

Description

Example

---

hh

IRIS serial date number for half-yearly date

Syntax

```
Dat = hh(Y)
Dat = hh(Y,H)
```

Input arguments

- Y [numeric] - Year.
- H [numeric] - Half-year; if missing, first half-year (January to June) is assumed.

Output arguments

- Dat [numeric] - IRIS serial date numbers representing the half-yearly date.

Description

Example

---

hhtoday

IRIS serial date number for current half-year

Syntax

```
Dat = hhtoday()
```

Output arguments

- Dat [numeric] - IRIS serial date number for current half-year.

Description

Example

---

mm

IRIS serial date number for monthly date

Syntax

```
Dat = mm(Y)
Dat = mm(Y,M)
```

Input arguments

- Y [numeric] - Year.
- M [numeric] - Month; if omitted, first month (January) is assumed.

Output arguments

- Dat [numeric] - IRIS serial date number representing the monthly date.

Description

Example

---

mmtoday

IRIS serial date number for current month

Syntax

```
Dat = mmtoday()
```

Output arguments

- `Dat` [numeric] - IRIS serial date number for current month.

Description

Example

---

qq

IRIS serial date number for quarterly date

Syntax

```
Dat = qq(Y)
Dat = qq(Y,Q)
```

Input arguments

- `Y` [numeric] - Year.
- `Q` [numeric] - Quarter; if omitted, first quarter is assumed.

Output arguments

- `Dat` [numeric] - IRIS serial date number representing the quarterly date.

Description

Example

---

qqtoday

IRIS serial date number for current quarter

Syntax

```
Dat = qqtoday()
```

Output arguments

- Dat [numeric] - IRIS serial date number for current quarter.

Description

Example

---

rngcmp

Compare two IRIS date ranges

Syntax

```
Flag = rngcmp(R1,R2)
```

Input arguments

- R1, R2 [numeric] - Two IRIS date ranges that will be compared.

Output arguments

- Flag [true | false] - True if the two date ranges are the same.

Description

An IRIS date range is distinct from a vector of dates in that only the first and the last dates matter. Often, date ranges are context sensitive. In that case, you can use -Inf for the start date (meaning the earliest possible date in the given context) and Inf for the end date (meaning the latest possible date in the given context), or simply Inf for the whole range (meaning from the earliest possible date to the latest possible date in the given context).

Example

```
r1 = qq(2010,1):qq(2020,4);
r2 = [qq(2010,1),qq(2020,4)];

rngcmp(r1,r2)
ans =
    1
```

■ str2dat

Convert strings to IRIS serial date numbers

Syntax

```
Dat = str2dat(S,...)
```

Input arguments

- S [char | cellstr] - Strings representing dates.

Output arguments

- Dat [numeric] - IRIS serial date numbers.

Options

- 'freq=' [1 | 2 | 4 | 6 | 12 | 52 | 365 | empty] - Enforce frequency.

See help on dat2str P280 for other options available.

Description

Example

```
d = str2dat('04-2010','dateFormat=','MM-YYYY');
dat2str(d)
ans =
   '2010M04'

d = str2dat('04-2010','dateFormat=','MM-YYYY','freq=',4);
dat2str(d)
ans =
   '2010Q2'
```

■ textinp2dat

Convert text input to IRIS serial date numbers

Syntax

```
Dat = textinp2dat(Str)
```

Input arguments

- `Str` [char] - String describing a date, a vector of dates, or a range; see Description.

Output arguments

- `Dat` [numeric] - IRIS serial date numbers representing the input date, vector of dates, or range.

Description

Input text strings can contain dates in the basic format, for instance `2010Y` for yearly dates, `2010H1` for half-yearly dates, `2010Q2` for quarterly dates, `2010B6` for bi-monthly dates, `2010M09` for monthly dates, `2010W52` for weekly dates, or `2010-May-30` for daily dates. Each occurrence of a date will be replaced with a call to the respective IRIS date function, `yy(...)`, `hh(...)`, `qq(...)`, `bb(...)`, `mm(...)`, `ww(...)`, or `dd(...)`, and the resulting expression will be evaluated, converting it into a vector of IRIS serial date numbers.

Example

```
>> textinp2dat('2010Q1:2011Q4')
ans =
   1.0e+03 *
    Columns 1 through 7
    8.0400    8.0410    8.0420    8.0430    8.0440    8.0450    8.0460
    Column 8
    8.0470
>> dat2str( textinp2dat('2010Q1:2011Q4') )
ans =
    Columns 1 through 6
    '2010Q1'    '2010Q2'    '2010Q3'    '2010Q4'    '2011Q1'    '2011Q2'
    Columns 7 through 8
    '2011Q3'    '2011Q4'
```

■ weeksinyear

Number of weeks in year

Syntax

```
N = weeksinyear(Year)
```

Input arguments

- `Year` [numeric] - Year.

Output arguments

- `N` [numeric] - Number of weeks in `Year`.

Description

The number of weeks in a year is either 52 or 53, and complies with the definition of the first week in a year in ISO 8601. The first week of a year is the one that contains the 4th day of January (in other words, has most of its days in that year).

Example

```
weeksinyear(2000:2010)
ans =
    52    52    52    52    53    52    52    52    52    53    52
```

---

ww

IRIS serial date number for weekly date

Syntax

```
Dat = ww(Year,Week)
Dat = ww(Year,Month,Day)
```

Input arguments

- `Year` [numeric] - Years.
- `Week` [numeric] - Week of the year.
- `Month` [numeric] - Calendar month.
- `Day` [numeric] - Calendar day of the month `Month`.

Output arguments

- `Dat` [numeric] - IRIS serial date number representing the weekly date.

Description

The IRIS weekly dates comply with the ISO 8601 definition:

- every week starts on Monday and ends on Sunday;
- the month or year to which the week belongs is determined by its Thurdsay.

Example

---

ww2day

Convert weekly IRIS serial date number to Matlab serial date number

Syntax

```
Day = ww2day(Dat)
Day = ww2day(Dat,WDay)
```

Input arguments

- `Dat` [numeric] - IRIS serial number for weekly date.
- `WDay` [`'Mon'` | `'Tue'` | `'Wed'` | `'Thu'` | `'Fri'` | `'Sat'` | `'Sun'`] - The day of the week that will represent the input week, `Dat`; if omitted, the week will be represented by its Thursday.

Output arguments

- `Day` [numeric] - Matlab serial date number representing Thursday in that week.

Description

Example

The first week of the year 2009 starts on Monday, 29 December 2008 (it is the first week of 2009 by ISO 8601 definition, because Thursday of that week falls in 2009).

The following command returns the Thursday of that week (note that `datestr` is a standard Matlab function, not an IRIS function),

```
firstWeek09 = ww(2009,1);
datestr( ww2day(firstWeek09) )
ans =
01-Jan-2009
```

while this command returns the Monday of the same week,

```
datestr( ww2day(firstWeek09,'Monday') )
ans =
29-Dec-2008
```

wwtoday

IRIS serial date number for current week

Syntax

```
Dat = wwtoday()
```

Output arguments

- Dat [numeric] - IRIS serial date number for current week.

Description

Example

---

yy

IRIS serial date number for yearly date

Syntax

```
Dat = yy(Y)
```

Input arguments

- Y [numeric] - Year.

Output arguments

- Dat [numeric] - IRIS serial date numbers representing the yearly date.

Description

Example

---

yytoday

IRIS serial date number for current year

Syntax

```
Dat = yytoday()
```

Output arguments

- Dat [numeric] - IRIS serial date number for current year.

Description

Example

18 Time Series (tseries Objects)

tseries methods:

Constructor

- tseries P375 - Create new time series (tseries) object.

Getting information about tseries objects

- enddate P331 - Date of the last available observation in a tseries object.
- freq P336 - Date frequency of tseries object.
- get P337 - Query tseries object property.
- isequal P343 - [Not a public function] Compare two tseries objects.
- length P344 - Length of tseries object.
- ndims P349 - Number of dimensions in tseries object data.
- size P366 - Size of tseries object data.
- specrange P367 - Time series specific range.
- startdate P370 - Date of the first available observation in a tseries object.
- yearly P382 - Display tseries object one calendar year per row.

Referencing tseries objects

- subsasgn P373 - Subscripted assignment for tseries objects.
- subsref P373 - Subscripted reference function for tseries objects.

Maths and statistics functions and operators

Some of the following functions require the Statistics Toolbox.

+, -, *, \, /, ^, &, |, ~, ==, ~=, >=, >, <, <=, abs, acos, asin, atan, atan2, ceil, cos, exp, fix, floor,imag, isinf, isnan, log, log10, real, round, sin, sqrt, tan, normpdf, normcdf, prctile, lognpdf, logncdf

The behaviour of the following functions depend on the dimension along which they are performed.

Some of the following functions require the Statistics Toolbox.

all, any, cumprod, cumsum, find, geomean, max, mean, median, min, mode, nanmean, nanstd, nansum, nanvar, prod, std, sum, var

Filters and evaluation

- arf P311 - Run autoregressive function on time series.
- arma P312 - Apply ARMA model to input series.
- bpass P318 - Band-pass filter.
- bwf P321 - Butterworth filter with tunes.
- bwf2 P323 - Swap output arguments of the Butterworth filter with tunes.
- detrend P328 - Remove a linear time trend.
- expsmooth P333 - Exponential smoothing.
- hpf P339 - Hodrick-Prescott filter with tunes (aka LRX filter).
- hpf2 P342 - Swap output arguments of the Hodrick-Prescott filter with tunes.
- fft P334 - Discrete Fourier transform of tseries object.
- llf P344 - Local level filter (aka random walk plus white noise) with tunes.
- llf2 P348 - Swap output arguments of the local linear trend filter with tunes.
- moving P348 - Apply function to moving window of observations.
- trend P374 - Estimate a time trend.
- x12 P377 - Access to X13-ARIMA-SEATS seasonal adjustment program.

Estimation and sample characteristics

Note that most of the sample characteristics are listed above in the Maths and statistics functions and operators section.

- acf P309 - Sample autocovariance and autocorrelation functions.
- hpdi P338 - Highest probability density interval.
- chowlin P323 - Chow-Lin distribution of low-frequency observations over higher-frequency periods.
- regress P359 - Ordinary or weighted least-square regression.

Visualising tseries objects

- area P310 - Area graph for tseries objects.
- band P314 - Line-and-band graph for tseries objects.
- bar P316 - Bar graph for tseries objects.
- barcon P317 - Contribution bar graph for tseries objects.
- bubble P320 - Bubble graph for tseries objects.
- errorbar P332 - Line plot with error bars.
- plot P352 - Line graph for tseries objects.
- plotcmp P353 - Comparison graph for two time series.
- plotpred P354 - Visualize multi-step-ahead predictions.
- plotyy P356 - Line plot function with LHS and RHS axes for time series.
- scatter P364 - Scatter graph for tseries objects.

- spy P369 - Visualise tseries observations that pass a test.
- stem P372 - Plot tseries as discrete sequence data.

Manipulating tseries objects

- empty P331 - Empty time series preserving the size in 2nd and higher dimensions.
- flipud P335 - Flip time series data up to down.
- permute P351 - Permute dimensions of a tseries object.
- repmat P360 - Repeat copies of time series data.
- redate P358 - Change time dimension of time series.
- reshape P361 - Reshape size of time series in 2nd and higher dimensions.
- resize P361 - Clip tseries object down to a specified date range.
- sort P367 - Sort tseries columns by specified criterion.

Converting tseries objects

- convert P324 - Convert tseries object to a different frequency.
- double P330 - Return tseries observations as double-precision numeric array.
- doubledata P330 - Convert tseries observations to double precision.
- single P365 - Return tseries observations as single-precision numeric array.
- singledata P365 - Convert tseries observations to single precision.

Other tseries functions

- apct P310 - Annualised percent rate of change.
- bsxfun P319 - Implement bsxfun for tseries class.
- cumsumk P326 - Cumulative sum with a k-period leap.
- destdise P328 - Destandardise tseries object by applying specified standard deviation and mean to it.
- diff P329 - First difference.
- interp P342 - Interpolate missing observations.
- normalise P349 - Normalise (or rebase) data to particular date.
- pct P350 - Percent rate of change.
- round P363 - Round tseries values to specified number of decimals.
- rmse P362 - Compute RMSE for given observations and predictions.
- stdise P371 - Standardise tseries data by subtracting mean and dividing by std deviation.
- windex P376 - Simple weighted or Divisia index.
- wmean P377 - Weighted average of time series observations.

Getting on-line help on tseries functions

```
help tseries
help tseries/function_name
```

Reference page for tseries

---

acf

Sample autocovariance and autocorrelation functions

Syntax

```
[C,R] = acf(X)
[C,R] = acf(X,Dates,...)
```

Input arguments

- `X` [tseries] - Tseries object.
- `Dates` [numeric | Inf] - Dates or date range from which the input tseries data will be used.

Output arguments

- `C` [numeric] - Auto-/cross-covariance matrices.
- `R` [numeric] - Auto-/cross-correlation matrices.

Options

- `'demean='` [`true` | `false`] - Remove mean from the data before computing the ACF.
- `'order='` [numeric | `0`] - Order up to which the ACF will be computed.
- `'smallSample='` [`true` | `false`] - Adjust degrees of freedom for small samples.

Description

Example

---

apct

Annualised percent rate of change

Syntax

```
X = apct(X)
```

Input arguments

- X [tseries] - Input tseries object.

Output arguments

- X [tseries] - Annualised percentage rate of change in the input data.

Description

Example

---

area

Area graph for tseries objects

Syntax

```
[H,Range] = area(X,...)
[H,Range] = area(Range,X,...)
[H,Range] = area(Ax,Range,X,...)
```

Input arguments

- Ax [handle | numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [tseries] - Input tseries object whose columns will be plotted as an area graph.

Output arguments

- H [handle | numeric] - Handles to areas plotted.
- Range [numeric] - Actually plotted date range.

Options

See help on tseries/plot P352 and the built-in function area for all options available.

Description

Example

---

arf

Run autoregressive function on time series

Syntax

```
X = arf(X,A,Z,Range,...)
```

Input arguments

- X [tseries] - Input data from which initial condition will be taken.
- A [numeric] - Vector of coefficients of the autoregressive polynomial.
- Z [numeric | tseries] - Exogenous input series or constant in the autoregressive process.
- Range [numeric | @all] - Date range on which the new time series observations will be computed; Range does not include pre-sample initial condition. @all means the entire possible range will be used (taking into account the length of pre-sample initial condition needed).

Output arguments

- X [tseries] - Output data with new observations created by running an autoregressive process described by A and Z.

Description

The autoregressive process has one of the following forms:

```
A1*x + A2*x(-1) + ... + An*x(-n) = z,
```

or

```
A1*x + A2*x(+1) + ... + An*x(+n) = z,
```

depending on whether the range is increasing (running forward in time), or decreasing (running backward in time). The coefficients A1,...An are gathered in the input vector A,

```
A = [A1,A2,...,An].
```

Example

The following two lines create an autoregressive process constructed from normally distributed residuals,

$$x_t = \rho x_{t-1} + \epsilon_t$$

```
rho = 0.8;
X = tseries(1:20,@randn);
X = arf(X,[1,-rho],X,2:20);
```

---

arma

Apply ARMA model to input series

Syntax

```
Y = arma(X,E,Ar,Ma,Range)
```

Input arguments

- X [tseries] - Input time series from which initial condition will be constructed.
- E [tseries] - Input time series with innovations; NaN values in E on Range will be replaced with 0.
- Ar [numeric | empty] - Row vector of AR polynominal coefficients; if empty, Ar = 1; see Description.
- Ma [numeric | empty] - Row vector of MA polynominal coefficients; if empty, Ma = 1; see Description.
- Range [numeric | char] - Range on which the output series observations will be constructed.

Output arguments

- X [tseries] - Output time series constructed by running an ARMA model on the input series X and E; the output time series also includes p initial conditions where p is the order of the AR polynomial.

Options

Description

The output series is constructed as follows:

$$A(L)X_t = M(L)E_t$$

where $A(L) = A_0 + A_1 L + \cdots$ and $M(L) = M_0 + M_1 L + \cdots$ are polynomials in lag operator L defined by the vectors Ar and Ma. In other words,

$$X_t = \frac{1}{A_1}\left(-A_2 X_{t-1} - A_3 X_{t-2} - \cdots + M_0 E_t + M_1 E_{t-1} + \cdots\right).$$

Note that the coefficient A_0 is Ar(1), A_1 is Ar(2), and so on.

Example

Construct an AR(1) process with autoregression coefficient 0.8, built from normally distributed innovations:

```
X = tseries(0:20,0);
E = tseries(1:20,@randn);
X = arma(X,E,[1,-0.8],[ ],1:20);
plot(X);
```

---

band

Line-and-band graph for tseries objects

Syntax

```
[Ln,Bd,Range] = band(X,Low,High...)
[Ln,Bd,Range] = band(Range,X,Low,High,...)
[Ln,Bd,Range] = band(Ax,Range,X,Low,High,...)
```

Input arguments

- `Ax` [numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- `Range` [numeric | char] - Date range; if not specified the entire range of the input time series object will be plotted.
- `X` [tseries] - Input time series whose columns will be plotted as a line graph (referred to as center lines).
- `Low` [tseries] - Time series that defines the lower edge of each band.
- `High` [tseries] - Time series that defines the upper edge of each band plotted.

Output arguments

- `Ln` [numeric] - Handles to lines plotted.
- `Bd` [numeric] - Handles to bands (patch objects) plotted.
- `Range` [numeric] - Date range actually plotted.

Options

- 'datePosition=' ['centre' | 'end' | 'start'] - Position of each date point within a given period span.
- 'dateTick=' [numeric | Inf] - Vector of dates locating tick marks on the X-axis; Inf means they will be created automatically.
- 'excludeFromLegend=' [*true* | false] - Excluce bands from legend.
- 'grid=' ['bottom' | 'top'] - Place grid on top or bottom.
- 'relative=' [true | false] - If true, the lower and upper edge will be constructed by subtracting Low from X and adding High to X, respectively; otherwise, Low and High will be interpreted as absolute positions of the edges.
- 'tight=' [true | false] - Make the y-axis tight.
- 'white=' [numeric | 0.85] - Percentage of white color mixed with the respective center line color and used to fill the band area.

See help on built-in plot function for other options available.

Date format options

See dat2str P280 for details on date format options.

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months.
- 'standinMonth=' [numeric | 'last' | 1] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.

Description

If one (or more) of the input time series, X, Low, or High, consists of more than one column, the graph is constructructed as follows:

- One column in X, multiple columns in Low or High - multiple bands are plotted around a single center line.
- Multiple columns in X, one column in Low or High - a single band is plotted around each of the center lines, each band constructed from the same lower and upper edge data; this setup makes sense only with the option 'relative=' true.
- Multiple columns in X, mutliple columns in Low or High - a single band is plotted around each of the center lines, each band constructed from different data.

Example

---

bar

Bar graph for tseries objects

Syntax

```
[H,Range] = bar(X,...)
[H,Range] = bar(Range,X,...)
[H,Range] = bar(Ax,Range,X,...)
```

Input arguments

- Ax [handle | numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date Range; if not specified the entire Range of the input tseries object will be plotted.
- X [tseries] - Input tseries object whose columns will be plotted as a bar graph.

Output arguments

- H [handle | numeric] - Handles to bars plotted.
- Range [numeric] - Actually plotted date Range.

Options

See help on tseries/bar P316 and the built-in function bar for all options available.

Description

Example

---

barcon

Contribution bar graph for tseries objects

Syntax

```
[H,Range] = barcon(X,...)
[H,Range] = barcon(Range,X,...)
[H,Range] = barcon(Ax,Range,X,...)
```

Input arguments

- Ax [handle | numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [tseries] - Input tseries object whose columns will be plotted as a contribution bar graph.

Output arguments

- H [handle | numeric] - Handles to the bars plotted.
- Range [numeric] - Actually plotted date range.

Options

- 'barWidth=' [numeric | 0.8] - Width of bars as a percentage of the space each period occupies on the x-axis.

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'colorMap=' [numeric | get(gcf(),'colorMap')] - Color map used to fill the contribution bars.
- 'evenlySpread=' [true | false] - Colors picked for the contribution bars are evenly spread across the color map.
- 'ordering=' ['ascend' | 'descend' | 'preserve' | numeric] - Ordering of contributions with the same sign withinin each period; 'preserve' means the original order will be preserved.

See help on tseries/plot P352 and the built-in function bar for other options available.

Description

Example

---

bpass

Band-pass filter

Syntax

```
[X,T] = bpass(X,Band,Range,...)
```

Output arguments

- X [tseries] - Band-pass filtered tseries object.
- T [tseries] - Estimated trend tseries object.

Input arguments

- X [tseries] - Input tseries object that will be filtered.
- Range [numeric | Inf] Date range on which the data will be filtered.
- Band [numeric] - Band of periodicities to be retained in the output data, Band = [LOW,HIGH].

Options

- 'addTrend=' [true | false] - Add the estimated linear time trend back to filtered output series if band includes Inf.
- 'detrend=' [true | false] - Remove an estimated time trend from the data before filtering.
- 'log=' [true | false] - Logarithmise the data before filtering, de-logarithmise afterwards.
- 'method=' ['cf' | 'hwfsf'] - Type of band-pass filter: Christiano-Fitzgerald, or h-windowed frequency-selective filter.
- 'unitRoot=' [true | false] - Assume unit root in the input data.

See help on tseries/trend P374 for other options available when 'detrend=' is set to true.

Description

Christiano, L.J. and T.J.Fitzgerald (2003). The Band Pass Filter. International Economic Review, 44(2), 435–465.

Iacobucci, A. & A. Noullez (2005). A Frequency Selective Filter for Short-Length Time Series. Computational Economics, 25, 75–102.

Example

---

bsxfun

Implement bsxfun for tseries class

Syntax

```
Z = bsxfun(Func,X,Y)
```

Input arguments

- Func [function_handle] - Function that will be applied to the input series, FUN(X,Y).
- X [tseries | numeric] - Input time series or numeric array.
- Y [tseries | numeric] - Input time series or numeric array.

Output arguments

- Z [tseries] - Result of Func(X,Y) with X and/or Y expanded properly in singleton dimensions.

Description

See help on built-in bsxfun for more help.

Example

Create a multivariate time series and subtract mean from its individual columns.

```
x = tseries(1:10,rand(10,4));
xx = bsxfun(@minus,x,mean(x));
```

---

bubble

Bubble graph for tseries objects

Syntax

```
[H1,H2,Range] = bubble([X,Y],...)
[H1,H2,Range] = bubble([X,Y,Z],...)
[H1,H2,Range] = bubble([X,Y,Z,C],...)
[H1,H2,Range] = bubble(Range,[X,Y],...)
[H1,H2,Range] = bubble(Range,[X,Y,Z],...)
[H1,H2,Range] = bubble(Range,[X,Y,Z,C],...)
[H1,H2,Range] = bubble(Ax,Range,[X,Y],...)
[H1,H2,Range] = bubble(Ax,Range,[X,Y,Z],...)
[H1,H2,Range] = bubble(Ax,Range,[X,Y,Z,C],...)
```

Input arguments

- Ax [handle | numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.

- `[X,Y,Z,C]` [tseries] - Requires the axes X and Y, and optionally accepts Z to control the size of the elements in the scatter plot, and optionally accepts C to control the colour.

Output arguments

- `H1` [handle | numeric] - Handles to scatter plot.
- `H2` [cell] - Cell array of handles to arrows.
- `Range` [numeric] - Actually plotted date range.

Options

See help on tseries/plot P352 for all options available.

Description

Example

---

bwf

Butterworth filter with tunes

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[T,C,CutOff,Lambda] = bwf(X,Order,~Range,...)
```

Syntax with output arguments swapped

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[T,C,CutOff,Lambda] = bwf2(X,Order,~Range,...)
```

Input arguments

- X [tseries] - Input tseries object that will be filtered.
- Order [numeric] - Order of the Butterworth filter; Order=2 reproduces the Hodrick-Prescott filter hpf P339, and Order=1 reproduces the local linear filter llf P344.
- ~Range [numeric | char | @all] - Date range on which the input data will be filtered; Range can be @all, Inf, [startdata,Inf], or [-Inf,enddate]; if omitted, @all (i.e. the entire available range of the input series) is used.

Output arguments

- T [tseries] - Lower-frequency (trend) component.
- C [tseries] - Higher-frequency (cyclical) component.
- CutOff [numeric] - Cut-off periodicity; periodicities above the cut-off are attributed to trends, periodicities below the cut-off are attributed to gaps.
- Lambda [numeric] - Smoothing parameter actually used; this output argument is useful when the option 'CutOff=' is used instead of 'Lambda='.

Options

- 'CutOff=' [numeric | empty] - Cut-off periodicity in periods (depending on the time series frequency); this option can be specified instead of 'Lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.
- 'CutOffYear=' [numeric | empty] - Cut-off periodicity in years; this option can be specified instead of 'Lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.

'infoSet=' [1 | 2] - Information set assumption used in the filter: 1 runs a one-sided filter, 2 runs a two-sided filter.

- 'Lambda=' [numeric | @auto] - Smoothing parameter; needs to be specified for tseries objects with indeterminate frequency. See Description for default values.
- 'level=' [tseries] - Time series with soft and hard tunes on the level of the trend.
- 'change=' [tseries] - Time series with soft and hard tunes on the change in the trend.
- 'log=' [true | false] - Logarithmise the data before filtering, de-logarithmise afterwards.

Description

Default smoothing parameters

If the user does not specify the smoothing parameter using the `'lambda='` option (or reassigns the default `@auto`), a default value is used. The default value is based on common practice and can be calculated using the date frequency of the input time series as $\lambda = (10 \cdot f)^n$, where f is the frequency (yearly=1, half-yearly=2, quarterly=4, bi-monthly=6, monthly=12), and n is the order of the filter, determined by the input parameter `Order`.

Example

---

bwf

Swap output arguments of the Butterworth filter with tunes

See help on tseries/bwf P321.

---

chowlin

Chow-Lin distribution of low-frequency observations over higher-frequency periods

Syntax

```
[Y2,B,RHO,U1,U2] = chowlin(Y1,X2)
[Y2,B,RHO,U1,U2] = chowlin(Y1,X2,Range,...)
```

Input arguments

- `Y1` [tseries] - Low-frequency input time series that will be distributed over higher-frequency observations.
- `X2` [tseries] - Time series with regressors used to distribute the input data.
- `Range` [numeric] - Low-frequency date range on which the distribution will be computed.

Output arguments

- Y2 [tseries] - Output data distributed with higher frequency.
- B [numeric] - Vector of regression coefficients.
- RHO [numeric] - Actually used autocorrelation coefficient in the residuals.
- U1 [tseries] - Low-frequency regression residuals.
- U2 [tseries] - Higher-frequency regression residuals.

Options

- 'constant=' [true | false] - Include a constant term in the regression.
- 'log=' [true | false] - Logarithmise the data before distribution, de-logarithmise afterwards.
- 'ngrid=' [numeric | 200] - Number of grid search points for finding autocorrelation coefficient for higher-frequency residuals.
- 'rho=' ['estimate' | 'positive' | 'negative' | numeric] - How to determine the autocorrelation coefficient for higher-frequency residuals.
- 'timeTrend=' [true | false] - Include a time trend in the regression.

Description

Chow,G.C., and A.Lin (1971). Best Linear Unbiased Interpolation, Distribution and Extrapolation of Time Series by Related Times Series. Review of Economics and Statistics, 53, pp. 372-75.

See also Appendix 2 in Robertson, J.C., and E.W.Tallman (1999). Vector Autoregressions: Forecasting and Reality. FRB Atlanta Economic Review, 1st Quarter 1999, pp.4-17.

Example

■ convert

Convert tseries object to a different frequency

Syntax

```
Y = convert(X,NewFreq,...)
Y = convert(X,NewFreq,Range,...)
```

Input arguments

- X [tseries] - Input tseries object that will be converted to a new frequency, freq, aggregating or intrapolating the data.
- NewFreq [numeric | char] - New frequency to which the input data will be converted: 1 or 'A' for yearly, 2 or 'H' for half-yearly, 4 or 'Q' for quarterly, 6 or 'B' for bi-monthly, and 12 or 'M' for monthly.
- Range [numeric] - Date range on which the input data will be converted.

Output arguments

- Y [tseries] - Output tseries created by converting X to the new frequency.

Options

- 'ignoreNaN=' [true | false] - Exclude NaNs from agreggation.
- 'missing=' [numeric | NaN | 'last'] - Replace missing observations with this value.

Options for high- to low-frequency conversion (aggregation)

- 'method=' [function_handle | 'first' | 'last' | @mean] - Method that will be used to aggregate the high frequency data.
- 'select=' [numeric | Inf] - Select only these high-frequency observations within each low-frequency period; Inf means all observations will be used.

Options for low- to high-frequency conversion (interpolation)

- 'method=' [char | 'cubic' | 'quadsum' | 'quadavg'] - Interpolation method; any option available in the built-in interp1 function can be used.
- 'position=' ['centre' | 'start' | 'end'] - Position of the low-frequency date grid.

Description

The function handle that you pass in through the 'method' option when you aggregate the data (convert higher frequency to lower frequency) should behave like the built-in functions mean, sum etc. In other words, it is expected to accept two input arguments:

- the data to be aggregated,
- the dimension along which the aggregation is calculated.

The function will be called with the second input argument set to 1, as the data are processed en block columnwise. If this call fails, convert will attempt to call the function with just one input argument, the data, but this is not a safe option under some circumstances since dimension mismatch may occur.

Example

---

cumsumk

Cumulative sum with a k-period leap

Syntax

```
Y = cumsumk(X,K,Rho,Range)
Y = cumsumk(X,K,Rho)
Y = cumsumk(X,K)
Y = cumsumk(X)
```

Input arguments

- X [tseries] - Input data.
- K [numeric] - Number of periods that will be leapt the cumulative sum will be taken; if not specified, K is chosen to match the frequency of the input data (e.g. K = -4 for quarterly data), or K = -1 for indeterminate frequency.
- Rho [numeric] - Autoregressive coefficient; if not specified, Rho = 1.
- Range [numeric] - Range on which the cumulative sum will be computed and the output series returned.

Output arguments

- Y [tseries] - Output data constructed as described below.

Options

- 'log=' [true | false] - Logarithmise the input data before, and de-logarithmise the output data back after, running x12.

Description

If K < 0, the first K observations in the output series Y are copied from X, and the new observations are given recursively by

```
Y{t} = Rho*Y{t-K} + X{t}.
```

If K > 0, the last K observations in the output series Y are copied from X, and the new observations are given recursively by

```
Y{t} = Rho*Y{t+K} + X{t},
```

going backwards in time.

If K == 0, the input data are returned.

Example

Construct random data with seasonal pattern, and run X12 to seasonally adjust these series.

```
x = tseries(qq(1990,1):qq(2020,4),@randn);
x1 = cumsumk(x,-4,1);
x2 = cumsumk(x,-4,0.7);
x1sa = x12(x1);
x2sa = x12(x2);
```

The new series x1 will be a unit-root process while x2 will be stationary. Note that the command on the second line could be replaced with x1 = cumsumk(x).

---

destdise

Destandardise tseries object by applying specified standard deviation and mean to it

Syntax

```
X = destdise(X,XMean,XStd)
```

Input arguments

- `X` [tseries] - Input tseries object.
- `XMean` [numeric] - Mean that will be added the data.
- `XStd` [numeric] - Standard deviation that will be added to the data.

Output arguments

- `X` [tseries] - Destandardised output data.

Description

Example

---

detrend

Remove a linear time trend

Syntax

```
X = detrend(X,...)
X = detrend(X,Range,...)
```

Input arguments

- X [tseries] - Input time series.
- Range [numeric | @all | char] - The date range on which the trend will be computed; @all means the entire range available will be used.

Output arguments

- x [tseries] - Output time series with a trend removed.

Options

See tseries/trend P374 for options available.

Description

Example

---

diff

First difference

Syntax

```
X = diff(X)
X = diff(X,K)
```

Input arguments

- X [tseries] - Input tseries object.
- K [numeric] - Number of periods over which the first difference will be computed; Y = X - X{K}. Note that K must be a negative number for the usual backward differencing. If not specified, K will be set to -1.

Output arguments

- X [tseries] - First difference of the input data.

Description

Example

■ double

Return tseries observations as double-precision numeric array

Syntax

```
y = double(x)
```

Input arguments

- x [tseries] - Tseries object whose observations will be returned as double-precision numeric array.

Output arguments

- y [numeric] - Double-precision numeric array with the input tseries observations in columns.

Description

Example

■ doubledata

Convert tseries observations to double precision

Syntax

```
x = doubledata(x)
```

Input arguments

- x [tseries] - Tseries object whose observations will be be converted to double precision.

Output arguments

- y [numeric] - Tseries object with double-precision observations.

Description

Example

---

empty

Empty time series preserving the size in 2nd and higher dimensions

Syntax

```
x = empty(x)
```

Input arguments

- This [tseries] - Input time series that will be emptied.

Output arguments

- This [tseries] - Empty time series with the 2nd and higher dimensions the same size as the input tseries object, and comments preserved.

Description

Example

---

enddate

Date of the last available observation in a tseries object

Syntax

```
D = enddate(X)
```

Input arguments

- `X` [tseries] - Tseries object.

Output arguments

- `D` [numeric] - IRIS serial date number representing the date of the last observation available in the input tseries.

Description

The `startdate` function is equivalent to calling

```
get(x,'endDate')
```

Example

---

errorbar

Line plot with error bars

Syntax

```
[LL,EE,Range] = errorbar(X,W,...)
[LL,EE,Range] = errorbar(Range,X,W,...)
[LL,EE,Range] = errorbar(AA,Range,X,W,...)
[LL,EE,Range] = errorbar(X,Lo,Hi,...)
[LL,EE,Range] = errorbar(Range,X,Lo,Hi,...)
[LL,EE,Range] = errorbar(AA,Range,X,Lo,Hi,...)
```

Input arguments

- AA [numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [tseries] - Tseries object whose data will be plotted as a line graph.
- W [tseries] - Width of the bands that will be plotted around the lines.
- Lo [tseries] - Width of the band below the line.
- Hi [tseries] - Width of the band above the line.

Output arguments

- LL [numeric] - Handles to lines plotted.
- EE [numeric] - Handles to error bars plotted.
- Range [numeric] - Actually plotted date range.

Options

- 'relative=' [true | false] - If true, the data for the lower and upper bounds are relative to the centre, i.e. the bounds will be added to the centre (in this case, Lo must be negative numbers and Hi must be positive numbers). If false, the bounds are absolute data (in this case Lo must be lower than X, and Hi must be higher than X).

See help on tseries/plot P352.

---

ews

Exponential smoothing

Syntax

```
X = expsmooth(X,Beta,...)
```

Input arguments

- X [tseries] - Input time series.
- Beta [numeric] - Exponential factor.

Output arguments

- X [tseries] - Exponentially smoothed series.

Options

- 'init=' [numeric | NaN] - Add this value before the first observation to initialise the smoothing.
- 'log=' [true | false] - Logarithmise the data before filtering, de-logarithmise afterwards.

Description

Examples

---

fft

Discrete Fourier transform of tseries object

Syntax

```
[y,range,freq,per] = fft(x)
[y,range,freq,per] = fft(x,range,...)
```

Input arguments

- x [tseries] - Input tseries object that will be transformed.
- range [numeric | Inf] - Date range.

Output arguments

- y [numeric] - Fourier transform with data organised in columns.
- range [numeric] - Actually used date range.
- freq [numeric] - Frequencies corresponding to FFT vector elements.
- per [numeric] - Periodicities corresponding to FFT vector elements.

Options

- 'full=' [true | false] - Return Fourier transform on the whole interval [0,2*pi]; if false only the interval [0,pi] is returned.

Description

Example

}

---

flipud

Flip time series data up to down

Syntax

```
X = flipud(X)
```

Input arguments

- X [tseries] - Time series whose data will be flipped up to down.

Output arguments

- X [tseries] - Time series with its data flipped up to down.

Description

The data vector or matrix of the input time series is flipped up to down using the standard Matlab function `flipud`, i.e. the rows of the data vector or matrix are reorganized from last to first.

Example

```
>> x = tseries(qq(2000,1):qq(2000,4),1:4)
x =
    tseries object: 4-by-1
    2000Q1:  1
    2000Q2:  2
    2000Q3:  3
    2000Q4:  4
    ''
    user data: empty
    export files: [0]
>> flipud(x)
ans =
    tseries object: 4-by-1
    2000Q1:  4
    2000Q2:  3
    2000Q3:  2
    2000Q4:  1
    ''
    user data: empty
    export files: [0]
```

---

freq

Date frequency of tseries object

Syntax

```
F = freq(X)
```

Input arguments

- `X` [tseries] - Input tseries object.

Output arguments

- F [0 | 1 | 2 | 4 | 6 | 12 | 52 | 365] - Date frequency of observations in the input tseries object; F is the number of periods within a year.

Description

The freq( ) function is equivalent to calling the get( ) function:

```
get(x,'freq')
```

Example

---

get

Query tseries object property

Syntax

```
Ans = get(X,Query)
[Ans,Ans,...] = get(X,Query,Query,...)
```

Input arguments

- X [model] - Tseries object.
- Query [char] - Query to the tseries object.

Output arguments

- Ans [...] - Answer to the query.

Valid queries to tseries objects

- 'end=' Returns [numeric] the date of the last observation.
- 'freq=' Returns [numeric] the frequency (periodicity) of the time series.

- 'nanEnd=' Returns [numeric] the last date at which observations are available in all columns; for scalar tseries, this query always returns the same as 'end'.
- 'nanRange=' Returns [numeric] the date range from 'nanstart' to 'nanend'; for scalar time series, this query always returns the same as 'range'.
- 'nanStart=' Returns [numeric] the first date at which observations are available in all columns; for scalar tseries, this query always returns the same as 'start'.
- 'range=' Returns [numeric] the date range from the first observation to the last observation.
- 'start=' Returns [numeric] the date of the first observation.

Description

---

hpdi

Highest probability density interval

Syntax

```
int = hpdi(x,prob)
```

Input arguments

- x [tseries] - Input data with random draws in each period.
- prob [numeric] - Percent coverage of the computed interval, between 0 and 100.

Output arguments

- int [tseries] - Output tseries object with two columns, i.e. lower bounds and upper bounds for each period.

Description

Example

---

hpf

Hodrick-Prescott filter with tunes (aka LRX filter)

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[T,C,CutOff,Lambda] = hpf(X,~Range,...)
```

Syntax with output arguments swapped

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[C,T,CutOff,Lambda] = hpf2(X,~Range,...)
```

Input arguments

- `X` [tseries] - Input tseries object that will be filtered.
- `~Range` [numeric | char | `@all`] - Date range on which the input data will be filtered; `Range` can be `@all`, `Inf`, `[startdata,Inf]`, or `[-Inf,enddate]`; if omitted, `@all` (i.e. the entire available range of the input series) is used.

Output arguments

- `T` [tseries] - Low-frequency (trend) component.
- `C` [tseries] - High-frequency (cyclical or gap) component.
- `CutOff` [numeric] - Cut-off periodicity; periodicities above the cut-off are attributed to trends, periodicities below the cut-off are attributed to gaps.
- `Lambda` [numeric] - Smoothing parameter actually used; this output argument is useful when the option `'cutoff='` is used instead of `'lambda='`.

Options

- 'cutoff=' [numeric | empty] - Cut-off periodicity in periods (depending on the time series frequency); this option can be specified instead of 'lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.
- 'cutoffYear=' [numeric | empty] - Cut-off periodicity in years; this option can be specified instead of 'lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.
- 'gamma=' [numeric | tseries | 1] - Weight or weights on the deviations of the trend from observations; it only makes sense to use this option to make the signal-to-noise ratio time-varying; see the optimisation problem below.

'infoSet=' [1 | 2] - Information set assumption used in the filter: 1 runs a one-sided filter, 2 runs a two-sided filter.

- 'lambda=' [numeric | @auto] - Smoothing parameter; needs to be specified for tseries objects with indeterminate frequency. See Description for default values.
- 'level=' [tseries] - Time series with hard tunes and soft tunes on the level of the trend.
- 'change=' [tseries] - Time series with hard tunes and soft tunes on the change in the trend.
- 'log=' [true | false] - Logarithmise the data before filtering, de-logarithmise afterwards.

Description

The underlying optimisation problem

The function hpf solves a constrained optimisation problem described by the following Lagrangian

$$\min_{\bar{y}_t,\omega_t,\sigma_t} \underbrace{\sum \lambda \left(\Delta \bar{y}_t - \Delta \bar{y}_{t-1}\right)^2 + \sum \gamma_t \left(\bar{y}_t - y_t\right)^2}_{\text{Plain HP with time-varying signal-to-noise ratio}} + \cdots$$

$$\cdots + \underbrace{\sum u_t \left(\bar{y}_t - a_t\right)^2}_{\text{Soft level tunes}} + \underbrace{\sum v_t \left(\Delta \bar{y}_t - b_t\right)^2}_{\text{Soft growth tunes}} + \underbrace{\sum \omega_t \left(\bar{y}_t - c_t\right)}_{\text{Hard level tunes}} + \underbrace{\sum \sigma_t \left(\Delta \bar{y}_t - d_t\right)}_{\text{Hard growth tunes}},$$

where

- Δ is the first-difference operator;
- λ is a (scalar) smoothing parameter;
- y_t are user-supplied observations;

- $\bar{y}_t$ is the fitted trend;
- γ_t are user-supplied weights to modify the basic signal-to-noise ratio over time (the default setting is $\gamma_t = 1$), entered in the option `'gamma='`;
- a_t and u_t are soft tunes on the level of the trend and the weights associated with these soft level tunes, respectively, entered together as complex numbers in the option `'level='`;
- b_t and v_t are soft tunes on the change in the level of the trend and the weights associated with these soft growth tunes, respectively, entered together as complex numbers in the option `'growth='`;
- c_t are hard tunes on the level of the trend, entered as real numbers in the option `'level='`;
- d_t are hard tunes on the change in the level of the trend, entered as real numbers in the option `'growth='`;
- ω_t are lagRange multipliers on the hard level tunes (note that these are computed as part of the optimisation problem, not entered by the user);
- σ_t are lagRange multipliers on the hard growth tunes (note that these are computed as part of the optimisation problem, not entered by the user).

Each of the summations in the above Lagrangian goes over those periods in which the respective bracketed terms are defined (observations or tunes exist). You can combine any number of any tunes in one run of `hpf`, including out-of-sample tunes (see below).

How to enter the tunes

- The hard tunes and soft tunes on the level of the trend are entered as time series through the option `'level='`.
- The hard tunes and soft tunes on the change in the trend are entered as time series through the option `'change='`.
- In the tseries objects entered through `'level='` and/or `'change='`, you can combine any number of hard and soft tune. In each particular period, you can obviously specify only a hard tune or only a soft tune. You can think of hard tunes as a special case of soft tunes with infinitely large weights.
- A hard tune is specified as a plain real number (i.e. a number with a zero complex part).
- A soft tune must be entered as a complex number whose real part specifies the tune itself, and the imaginary part specifies the inverse of the weight, i.e. $1/v_t$ or $1/u_t$, on that tune in that period. Note that if the weight goes to infinity, the imaginary part becomes zero and the tune becomes a hard tune.

Out-of-sample tunes

Tunes can be imposed also at dates before the first observation of the input series, or after the last observation. In other words, the time series in `'level='` and/or `'growth='` can have a more extended Range (at either side) than the filtered input series.

Default smoothing parameters

If the user does not specify the smoothing parameter using the 'lambda=' option (or reassigns the default @auto), a default value is used. The default value is based on common practice and can be calculated using the date frequency of the input time series as $\lambda = 100 \cdot f^2$, where f is the frequency (yearly=1, half-yearly=2, quarterly=4, bi-monthly=6, monthly=12). This gives the following default values:

- 100 for yearly time series (cut-off periodicity of 19.79 years);
- 400 for half-yearly time series (cut-off periodicity of 14.02 years);
- 1,600 for quarterly time series (cut-off periodicity of 9.92 years);
- 3,600 for bi-monthly time series (cut-off periodicity of 8.11 years);
- 14,400 for monthly time series (cut-off periodicity of 5.73 years).

Note that there is no default value for data with indeterminate or daily frequency: for these types of time series, you must always use the option "lambda=".

Example

---

hpf2

Swap output arguments of the Hodrick-Prescott filter with tunes

See help on tseries/hpf P339.

---

interp

Interpolate missing observations

Syntax

```
X = interp(X,Range,...)
```

Input arguments

- X [tseries] - Input time series.

- Range [numeric | char] - Date range on which any missing observations (NaN) will be interpolated.

Output arguments

- x [tseries] - Tseries object with the missing observations interpolated.

Options

- 'method=' [char | 'cubic'] - Any valid method accepted by the built-in interp1 function.

Description

Example

---

isequal

[Not a public function] Compare two tseries objects

Syntax

```
Flag = isequal(X1,X2)
```

Input arguments

- X1, X2 [tseries] - Two tseries objects that will be compared.

Output arguments

- Flag [true | false] - True if the two input tseries objects have identical contents: start date, data, comments, userdata, and captions.

Description

The function isequaln is used to compare the tseries data, i.e. NaNs are correctly matched.

Example

---

length

Length of tseries object

Syntax

```
n = length(x)
```

Input arguments

- x [tseries] Tseries object.

Output arguments

- n [numeric] - Number of periods from the first to the last available observation in the input tseries object.

Description

Example

---

llf

Local level filter (aka random walk plus white noise) with tunes

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[T,C,CutOff,Lambda] = llf(X,~Range,...)
```

Syntax with output arguments swapped

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[C,T,CutOff,Lambda] = llf2(X,~Range,...)
```

Input arguments

- X [tseries] - Input tseries object that will be filtered.
- ~Range [numeric | char | @all] - Date range on which the input data will be filtered; Range can be @all, Inf, [startdata,Inf], or [-Inf,enddate]; if omitted, @all (i.e. the entire available range of the input series) is used.

Output arguments

- T [tseries] - Low-frequency (trend) component.
- C [tseries] - High-frequency (cyclical or gap) component.
- CutOff [numeric] - Cut-off periodicity; periodicities above the cut-off are attributed to trends, periodicities below the cut-off are attributed to gaps.
- Lambda [numeric] - Smoothing parameter actually used; this output argument is useful when the option 'cutoff=' is used instead of 'lambda='.

Options

- 'cutoff=' [numeric | empty] - Cut-off periodicity in periods (depending on the time series frequency); this option can be specified instead of 'lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.
- 'cutoffYear=' [numeric | empty] - Cut-off periodicity in years; this option can be specified instead of 'lambda='; the smoothing parameter will be then determined based on the cut-off periodicity.
- 'gamma=' [numeric | tseries | 1] - Weight or weights on the deviations of the trend from observations; it only makes sense to use this option to make the signal-to-noise ratio time-varying; see the optimisation problem below.
- 'drift=' [numeric | tseries | 0] - Deterministic drift in the trend.

`'infoSet='` [1 | 2] - Information set assumption used in the filter: 1 runs a one-sided filter, 2 runs a two-sided filter.

- `'lambda='` [numeric | `@auto`] - Smoothing parameter; needs to be specified for tseries objects with indeterminate frequency. See Description for default values.
- `'level='` [tseries] - Time series with soft and hard tunes on the level of the trend.
- `'change='` [tseries] - Time series with soft and hard tunes on the change in the trend.
- `'log='` [`true` | `false`] - Logarithmise the data before filtering, de-logarithmise afterwards.

Description

The underlying optimisation problem

The function `llf` solves a constrained optimisation problem described by the following Lagrangian

$$\min_{\bar{y}_t,\omega_t,\sigma_t} \underbrace{\sum \lambda \left(\Delta \bar{y}_t - \delta_t\right)^2 + \sum \gamma_t \left(\bar{y}_t - y_t\right)^2}_{\text{Plain local level filter with time-varying signal-to-noise ratio}} + \cdots$$

$$\cdots + \underbrace{\sum u_t \left(\bar{y}_t - a_t\right)^2}_{\text{Soft level tunes}} + \underbrace{\sum v_t \left(\Delta \bar{y}_t - b_t\right)^2}_{\text{Soft growth tunes}} + \underbrace{\sum \omega_t \left(\bar{y}_t - c_t\right)}_{\text{Hard level tunes}} + \underbrace{\sum \sigma_t \left(\Delta \bar{y}_t - d_t\right)}_{\text{Hard growth tunes}},$$

where

- Δ is the first-difference operator;
- λ is a (scalar) smoothing parameter;
- y_t are user-supplied observations;
- $\bar{y}_t$ is the fitted trend;
- δ_t is a user-supplied drift, either constant or time-varying, enetered in the option `'drift='`;
- γ_t are user-supplied weights to modify the basic signal-to-noise ratio over time (the default setting is $\gamma_t = 1$), entered in the option `'gamma='`;
- a_t and u_t are soft tunes on the level of the trend and the weights associated with these soft level tunes, respectively, entered together as complex numbers in the option `'level='`;
- b_t and v_t are soft tunes on the change in the level of the trend and the weights associated with these soft growth tunes, respectively, entered together as complex numbers in the option `'growth='`;
- c_t are hard tunes on the level of the trend, entered as real numbers in the option `'level='`;
- d_t are hard tunes on the change in the level of the trend, entered as real numbers in the option `'growth='`;

- ω_t are lagrange multipliers on the hard level tunes (note that these are computed as part of the optimisation problem, not entered by the user);
- σ_t are lagrange multipliers on the hard growth tunes (note that these are computed as part of the optimisation problem, not entered by the user).

Each of the summations in the above Lagrangian goes over those periods in which the respective bracketed terms are defined (observations or tunes exist). You can combine any number of any tunes in one run of `llf`, including out-of-sample tunes (see below).

How to enter the tunes

- The soft and hard tunes on the level of the trend are entered as time series through the option `'level='`.
- The soft and hard tunes on the change in the trend are entered as time series through the option `'change='`.
- In the tseries objects entered through `'level='` and/or `'change='`, you can combine any number of hard and soft tune. In each particular period, you can obviously specify only a hard tune or only a soft tune. You can think of hard tunes as a special case of soft tunes with infinitely large weights.
- A hard tune is specified as a plain real number (i.e. a number with a zero complex part).
- A soft tune must be entered as a complex number whose real part specifies the tune itself, and the imaginary part specifies the inverse of the weight, i.e. $1/v_t$ or $1/u_t$, on that tune in that period. Note that if the weight goes to infinity, the imaginary part becomes zero and the tune becomes a hard tune.

Out-of-sample tunes

Tunes can be imposed also at dates before the first observation of the input series, or after the last observation. In other words, the time series in `'level='` and/or `'growth='` can have a more extended range (at either side) than the filtered input series.

Default smoothing parameters

If the user does not specify the smoothing parameter using the `'lambda='` option (or reassigns the default `@auto`), a default value is used. The default value is based on common practice and can be calculated using the date frequency of the input time series as `\lambda = 10 \cdot f`, where `f` is the frequency (yearly=1, half-yearly=2, quarterly=4, bi-monthly=6, monthly=12). This gives the following default values:

- 10 for yearly time series (cut-off periodicity of 19.79 years);
- 20 for half-yearly time series (cut-off periodicity of 14.02 years);

- 40 for quarterly time series (cut-off periodicity of 9.92 years);
- 60 for bi-monthly time series (cut-off periodicity of 8.11 years);
- 120 for monthly time series (cut-off periodicity of 5.73 years).

Note that there is no default value for data with indeterminate or daily frequency: for these types of time series, you must always use the option "lambda=".

Example

■ llf2

Swap output arguments of the local linear trend filter with tunes

See help on tseries/llf P344.

■ moving

Apply function to moving window of observations

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
X = moving(X,~Range,...)
```

Input arguments

- X [tseries] - Tseries object on whose observations the function will be applied.
- ~Range [numeric | char | @all] - Date range from which input time series date will be used; @all means the entire range on which the input time series X is defined.

Output arguments

- X [tseries] - Output time series.

Options

- 'function=' [function_handle | @mean] - Function to be applied to moving window of observations.
- 'window=' [numeric | @auto] - The window of observations where 0 means the current date, -1 means one period lag, etc. @auto means that the last N observations (including the current one) are used, where N is the frequency of the input data.

Description

Example

---

ndims

Number of dimensions in tseries object data

Syntax

```
N = ndims(X)
```

Input arguments

- X [tseries] - Input tseries object.

Output arguments

- N [numeric] - Number of dimensions in the input object.

Description

Example

---

normalise

Normalise (or rebase) data to particular date

Syntax

```
X = normalise(X,NormDate,...)
```

Input arguments

- `x` [tseries] - Input time series that will be normalised.
- `NormDate` [numeric | `'start'` | `'end'` | `'nanStart'` | `'nanEnd'`] - Date relative to which the input data will be normalised; if not specified, `'nanStart'` (the first date for which all columns have an observation) will be used.

Output arguments

- `X` [tseries] - Normalised time series.

Options

- `'mode='` [`'add'` | `'mult'`] - Additive or multiplicative normalisation.

Description

Example

---

pct

Percent rate of change

Syntax

```
X = pct(X)
X = pct(X,K,...)
```

Input arguments

- `X` [tseries] - Input tseries object.
- `K` [numeric] - Time shift over which the rate of change will be computed, i.e. between time t and t+k; if not specified `K` will be set to `-1`.

Output arguments

- X [tseries] - Percentage rate of change in the input data.

Options

- 'outputFreq=' [1 | 2 | 4 | 6 | 12 | empty] - Convert the rate of change to the requested date frequency; empty means plain rate of change with no conversion.

Description

Example

In this example, x is a monthly time series. The following command computes the annualised rate of change between month t and t-1:

```
pct(x,-1,'outputfreq=',1)
```

while the following line computes the annualised rate of change between month t and t-3:

```
pct(x,-3,'outputFreq=',1)
```

---

permute

Permute dimensions of a tseries object

Syntax

```
X = permute(X,Order)
```

Input arguments

- X [tseries] - Tseries object whose dimensions, except the first (time) dimension, will be rearranged in the order specified by the vector order.
- Order [numeric] - New order of dimensions; because the time dimension cannot be permuted, order(1) must be always 1.

Output arguments

- X [tseries] - Output tseries object with its dimensions permuted.

Description

See help on the standard Matlab function permute.

Example

---

plot

Line graph for tseries objects

Syntax

```
[H,Range] = plot(X,...)
[H,Range] = plot(Range,X,...)
[H,Range] = plot(Ax,Range,X,...)
```

Input arguments

- Ax [numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [tseries] - Input tseries object whose columns will be plotted as a line graph.

Output arguments

- H [numeric] - Handles to lines plotted.
- Range [numeric] - Actually plotted date range.

Options

- 'datePosition=' ['centre' | 'end' | 'start'] - Position of each date point within a given period span.
- 'dateTick=' [numeric | Inf] - Vector of dates locating tick marks on the X-axis; Inf means they will be created automatically.
- 'tight=' [true | false] - Make the y-axis tight.

See help on built-in plot function for other options available.

Date format options

See dat2str P280 for details on date format options.

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months.
- 'standinMonth=' [numeric | 'last' | 1] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.

Description

Example

■ plotcmp

Comparison graph for two time series

Syntax

```
[Ax,Lhs,Rhs] = plotcmp(X,...)
[Ax,Lhs,Rhs] = plotcmp(Range,X,...)
```

Input arguments

- Range [numeric] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [tseries] - Tseries object with two or more columns; the difference (between the second and the first column (or any other linear combination of its columns specified through the option 'compare=') will be displayed as an RHS area or bar graph.

Output arguments

- Ax [handle | numeric] - Handles to the LHS and RHS axes.
- Lhs [handle | numeric] - Handles to the two original lines.
- Rhs [handle | numeric] - Handles to the area or bar difference graph.

Options

- 'baseLine=' [true | false] - Draw a baseline in the bar/area difference graph.
- 'compare=' [numeric | [-1;1]] - Linear combination of the observations that will be plotted in the RHS graph; [-1;1] means a difference between the second series and the first series, X{:,2}-X{:,1}.
- 'cmpColor=' [numeric | [1,0.75,0.75]] - Color that will be used to plot the area or bar difference (comparison) graph.
- 'cmpPlotFunc=' [@area | @bar] - Function that will be used to plot the difference (comparision) data on the RHS.

See help on tseries/plotyy P356 for other options available.

Description

Example

---

plotpred

Visualize multi-step-ahead predictions

Syntax

```
[H1,H2,H3] = plotpred(X,Y,...)
[H1,H2,H3] = plotpred(Ax,X,Y,...)
[H1,H2,H3] = plotpred(Ax,Range,X,Y,...)
```

Input arguments

- X [tseries] - Input data with time series observations.
- Y [tseries] - Prediction data arranged as described below; the prediction data returned from a Kalman filter can be used, see Example below.
- Ax [numeric] - Handle to axes object in which the data will be plotted.
- Range [numeric | Inf] - Date range on which the input data will be plotted.

Output arguments

- H1 [numeric] - Handles to a line object showing the time series observations (the first column, X, in the input data).
- H2 [numeric] - Handles to line objects showing the Kalman filter predictions (the second and further columns, Y, in the input data).
- H3 [numeric] - Handles to one-point line objects displaying a marker at the start of each line.

Options

- 'connect=' [true | false] - Connect the prediction lines, Y, with the corresponding observation in X.
- 'firstMarker=' ['none' | char] - Type of marker displayed at the start of each prediction line.
- 'showNaNLines=' [true | false] - Show or remove lines with whose starting points are NaN (missing observations).

See help on plot P352 and on the built-in function plot for options available.

Description

The input data Y need to be a multicolumn time series (tseries object), with one-step-ahead predictions x(t|t-1) in the first column, two-step-ahead predictions x(t|t-2) in the second column, and so on. Note the timing assumptions.

If x1 is a series with one-step-ahead predictions x(t+1|t), x2 is a series with two-step-ahead predictions x(t+2|t), and so on, while x is a series with the actual observations x(t), the following command will create a time series that can be then passed into plotpred( ):

```
p = [ x1{-1}, x2{-2}, ..., xn{-n} ];
plotpred(x, p);
```

Example

The plotpred( ) function can be used with prediction-step data returned from a Kalman filter, filter P93. The prediction-step data need to be specifically requested using the 'output=' option (as they are not included in the output database by default), with the prediction horizon assigned in the 'ahead=' option (the horizon is 1 by default):

```
[~, g] = filter(m, d, startDate:endDate, ...
    'output=', 'pred', 'meanOnly=', true, 'ahead=', 8);

figure( );
plotpred(startdate:enddate, d.x, g.pred.x);
```

---

plotyy

Line plot function with LHS and RHS axes for time series

Syntax

```
[Ax,Lhs,Rhs,Range] = plotyy(X,Y,...)
[Ax,Lhs,Rhs,Range] = plotyy(Range,X,Y,...)
[Ax,Lhs,Rhs,Range] = plotyy(RangeLhs,X,RangeRhs,Y,...)
```

Input arguments

- Range [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- RangeLhs [numeric | char] - LHS plot date range.
- RangeRhs [numeric | char] - RHS plot date range.
- X [tseries] - Input tseries object whose columns will be plotted and labelled on the LHS.
- Y [tseries] - Input tseries object whose columns will be plotted and labelled on the RHS.

Output arguments

- Ax [handle | numeric] - Handles to the LHS and RHS axes.
- Lhs [handle | numeric] - Handles to series plotted on the LHS axis.
- Rhs [handle | numeric] - Handles to series plotted on the RHS axis.
- Range [handle | numeric] - Actually plotted date range.

Options

- 'coincide=' [true | false] - Make the LHS and RHS y-axis grids coincide.
- 'lhsPlotFunc=' [@area | @bar | @plot | @stem] - Function that will be used to plot the LHS data.
- 'lhsTight=' [true | false] - Make the LHS y-axis tight.
- 'rhsPlotFunc=' [@area | @bar | @plot | @stem] - Function that will be used to plot the RHS data.
- 'rhsTight=' [true | false] - Make the RHS y-axis tight.

See help on tseries/plot P352 and the built-in function plotyy for all options available.

Description

Example

---

redate

Change time dimension of time series

Syntax

```
X = redate(X,oldDate,newDate)
```

Input arguments

- X [tseries] - Input time series.
- OldDate [numeric] - Base date that will be converted to a new date; OldDate does not need to be the stard date of X and does not even need to be within the current date range of X.
- NewDate [numeric] - A new date to which the base date oldDate will be changed; NewDate need not be the same frequency as OldDate.

Output arguments

- X [tseries] - Output tseries object with identical data as the input tseries object, but with its time dimension changed.

Description

Example

Create a time series on a date range from 2000Q1 to 2000Q4. Change the time dimension of the time series so that 1999Q4 (which is a date outside the original time series range) changes into 2009Q4 (which will again be a date outside the new time series range).

```
>> x = tseries(qq(2000,1):qq(2000,4),1:4)
x =
    tseries object: 4-by-1
    2000Q1:  1
    2000Q2:  2
    2000Q3:  3
    2000Q4:  4
    ''
    user data: empty
    export files: [0]
```

```
>> redate(x,qq(1999,4),qq(2009,4))
ans =
    tseries object: 4-by-1
    2010Q1:  1
    2010Q2:  2
    2010Q3:  3
    2010Q4:  4
    ''
    user data: empty
    export files: [0]
```

---

regress

Ordinary or weighted least-square regression

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[B,BStd,E,EStd,YFit,Range,BCov] = regress(Y,X,~Range,...)
```

Input arguments

- `Y` [tseries] - Tseries object with independent (LHS) variables.
- `X` [tseries] - Tseries object with regressors (RHS) variables.
- `~Range` [numeric] - Date range on which the regression will be run; if omitted, the entire range available will be used.

Output arguments

- `B` [numeric] - Vector of estimated regression coefficients.
- `BStd` [numeric] - Vector of std errors of the estimates.
- `E` [tseries] - Tseries object with the regression residuals.
- `EStd` [numeric] - Estimate of the std deviation of the regression residuals.

- YFit [tseries] - Tseries object with fitted LHS variables.
- Range [numeric] - The actually used date range.
- bBCov [numeric] - Covariance matrix of the coefficient estimates.

Options

- 'constant=' [true | false] - Include a constant vector in the regression; if true the constant will be placed last in the matrix of regressors.
- 'weighting=' [tseries | empty] - Tseries object with weights on observations in individual periods.

Description

This function calls the built-in lscov function.

Example

---

repmat

Repeat copies of time series data

Syntax

```
X = repmat(X,Rep1,Rep2,...)
```

Input arguments

- X [tseries] - Input time series.
- Rep1, Rep2, ... [numeric] - List of scalars that describe how copies of X data are arranged in each dimension.

Output arguments

- X [tseries] - Output time series.

Description

See help on built-in bsxfun for more help.

Example

---

reshape

Reshape size of time series in 2nd and higher dimensions

Syntax

```
x = reshape(x,newsize)
```

Input arguments

- x [tseries] - Tseries object whose data will be reshaped in 2nd and/or higher dimensions.
- newsize [numeric] - New size of the tseries object data; the first dimension (time) must be preserved.

Output arguments

- x [tseries] - Reshaped tseries object.

Description

Example

---

resize

Clip tseries object down to a specified date range

Syntax

```
X = resize(X,Range)
```

Input arguments

- X [tseries] - Input tseries object whose date range will be clipped down.
- Range [numeric] - New date range to which the input tseries object will be resized; the range can be specified as a [startDate,endDate] vector where -Inf and Inf can be used for the dates.

Output arguments

- X [tseries] - Output tseries object with its date range clipped down to Range.

Description

Example

---

rmse

Compute RMSE for given observations and predictions

Syntax

```
[Rmse,Pe] = rmse(Obs,Pred)
[Rmse,Pe] = rmse(Obs,Pred,Range,...)
```

Input arguments

- Obs [tseries] - Input data with observations.
- Pred [tseries] - Input data with predictions (a different prediction horizon in each column); Pred is typically the outcome of the Kalman filter, model/filter P93 or VAR/filter P221, called with the option 'ahead='.
- Range [numeric | Inf] - Date range on which the RMSEs will be evaluated; Inf means the entire possible range available.

Output arguments

- Rmse [numeric] - Numeric array with RMSEs for each column of Pred.
- Pe [tseries] - Prediction errors, i.e. the difference Obs - Pred evaluated within Range.

Description

Example

---

round

Round tseries values to specified number of decimals

Syntax

```
X = round(X)
X = round(X,Dec)
X = round(X,Dec,'significant')
```

Input arguments

- X [tseries] - Tseries object whose data will be rounded.
- Dec [numeric] - Number of decimals to which the tseries data will be rounded; if not specified, the data are rounded to nearest integer.
- 'significant' - See documentation on the built-in Matlab function round; works only in R2014b or later.

Output arguments

- X [tseries] - Rounded tseries object.

Description

The number of decimals, to which the tseries data will be rounded, can be positive, zero, or negative.

Example

---

scatter

Scatter graph for tseries objects

Syntax

```
[H,Range] = scatter([X,Y],...)
[H,Range] = scatter([X,Y,Z],...)
[H,Range] = scatter([X,Y,Z,C],...)
[H,Range] = scatter(Range,[X,Y],...)
[H,Range] = scatter(Range,[X,Y,Z],...)
[H,Range] = scatter(Range,[X,Y,Z,C],...)
[H,Range] = scatter(Ax,Range,[X,Y],...)
[H,Range] = scatter(Ax,Range,[X,Y,Z],...)
[H,Range] = scatter(Ax,Range,[X,Y,Z,C],...)
```

Input arguments

- `Ax` [numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- `Range` [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- `[X, Y, Z, C]` [tseries] - Requires the axes X and Y, and optionally accepts Z to control the size of the elements, and optionally accepts C to control the colour.

Output arguments

- `H` [numeric] - Handles to the lines plotted.
- `Range` [numeric] - Actually plotted date range.

Options

See help on tseries/plot P352 and the built-in function `scatter` for all options available.

Description

Example

■ single

Return tseries observations as single-precision numeric array

Syntax

```
y = single(x)
```

Input arguments

- x [tseries] - Tseries object whose observations will be returned as single-precision numeric array.

Output arguments

- y [numeric] - Single-precision numeric array with the input tseries observations in columns.

Description

Example

■ singledata

Convert tseries observations to single precision

Syntax

```
x = singledata(x)
```

Input arguments

- x [tseries] - Tseries object whose observations will be be converted to single precision.

Output arguments

- y [numeric] - Tseries object with single-precision observations.

Description

Example

---

size

Size of tseries object data

Syntax

```
S = size(X)
[S1,S2,...,Sn] = size(X)
```

Input arguments

- X [tseries] - Tseries object whose size will be returned.

Output arguments

- S [numeric] - Vector of sizes of the tseries object data in each dimension, S = [S1,S2,...,Sn].
- S1, S2, ..., Sn [numeric] - Sizes of the tseries object data in each dimension.

Description

Example

---

sort

Sort tseries columns by specified criterion

Syntax

```
[Y,INDEX] = sort(X,CRIT)
```

Input arguments

- `X` [tseries] - Input tseries object whose columns will be sorted in order determined by the criterion `crit`.
- `CRIT` ['sumsq' | 'sumabs' | 'max' | 'maxabs' | 'min' | 'minabs'] - Criterion used to sort the input tseries object columns.

Output arguments

- `Y` [tseries] - Output tseries object with columns sorted in order determined by the input criterion, `CRIT`.
- `INDEX` [numeric] - Vector of indices, `y = x{:,index}`.

Description

Example

---

specrange

Time series specific range

Syntax

```
Rng = specrange(X,S)
```

Input arguments

- X [tseries] - Time series.
- S [numeric | @all] - Range specification; the output range Rng will be constructed from the first and the last element of S only.

Output arguments

- Rng [numeric] - Date range constructed from S specific to time series X.

Description

The time series specific range is constructed as startDate:endDate where

- the start date startDate is S(1) if S(1) is a serial date number, or the start date of the input series X if S(1) is Inf, -Inf, or @all;
- the end date endDate is S(end) if S(end) is a serial date number, or the end date of the input series X if S(end) is Inf, or @all.

Example

Create a time series from 2000Q1 to 2001Q4

```
>> x = tseries( qq(2000,1):qq(2001,4), @rand );
```

The function specrange returns the full range of the time series when S is Inf

```
>> dat2str( specrange(x,Inf) )
ans =
    Columns 1 through 6
    '2000Q1'    '2000Q2'    '2000Q3'    '2000Q4'    '2001Q1'    '2001Q2'
    Columns 7 through 8
    '2001Q3'    '2001Q4'
```

or when S is [-Inf,Inf]

```
>> dat2str( specrange(x,[-Inf,Inf]) )
ans =
    Columns 1 through 6
```

```
    '2000Q1'    '2000Q2'    '2000Q3'    '2000Q4'    '2001Q1'    '2001Q2'
    Columns 7 through 8
    '2001Q3'    '2001Q4'
```

or when S is @all

```
>> dat2str( specrange(x,@all) )
ans =
    Columns 1 through 6
    '2000Q1'    '2000Q2'    '2000Q3'    '2000Q4'    '2001Q1'    '2001Q2'
    Columns 7 through 8
    '2001Q3'    '2001Q4'
```

A range from the start of the time series to a specific date is returned when S(1) is -Inf and S(end) is that specific end date:

```
>> dat2str( specrange(x,[-Inf,qq(2000,3)]) )
ans =
    '2000Q1'    '2000Q2'    '2000Q3'
```

A range from a specific date to the end of the time series is returned when S(1) is that specific start date date, and S(end) is Inf:

```
>> dat2str( specrange(x,[qq(2000,3),Inf]) )
ans =
    '2000Q3'    '2000Q4'    '2001Q1'    '2001Q2'    '2001Q3'    '2001Q4'
```

---

spy

Visualise tseries observations that pass a test

Syntax

```
[AA,LL] = spy(X,...)
[AA,LL] = spy(RANGE,X,...)
```

Input arguments

- X [tseries] - Input tseries object whose non-NaN observations will be plotted as markers.
- RANGE [tseries] - Date range on which the tseries observations will be visualised; if not specified the entire available range will be used.

Output arguments

- AA [tseries] - Handle to the axes created.
- LL [tseries] - Handle to the marks plotted.

Options

- 'names=' [cellstr] - Names that will be used to annotate individual columns of the input tseries object.
- 'test=' [function_handle | @(x)~isnan(x)] - Test applied to each observations; only the values returning a true will be displayed.

See help on tseries/plot P352 and the built-in function spy for all options available.

Description

Example

---

startdate

Date of the first available observation in a tseries object

Syntax

```
D = startdate(X)
```

Input arguments

- X [tseries] - Tseries object.

Output arguments

- D [numeric] - IRIS serial date number representing the date of the first observation available in the input tseries.

Description

The startdate function is equivalent to calling

```
get(X,'startDate')
```

Example

---

stdise

Standardise tseries data by subtracting mean and dividing by std deviation

Syntax

```
[X,M,S] = stdise(X)
[X,M,S] = stdise(X,Flag)
```

Input arguments

- X [tseries] - Input tseries object whose data will be normalised.
- Flag [0 | 1] - flag==0 normalises by N-1, flag==1 normalises by N, where N is the sample length.

Output arguments

- X [tseries] - Output tseries object with standardised data.
- XMeam [numeric] - Estimated mean subtracted from the input tseries observations.
- XStd [numeric] - Estimated std deviation by which the input tseries observations have been divided.

Description

Example

---

stem

Plot tseries as discrete sequence data

Syntax

```
[H,Range] = stem(X,...)
[H,Range] = stem(Range,X,...)
[H,Range] = stem(Ax,Range,X,...)
```

Input arguments

- `Ax` [handle | numeric] - Handle to axes in which the graph will be plotted; if not specified, the current axes will used.
- `Range` [numeric | char] - Date range; if not specified the entire range of the input tseries object will be plotted.
- `X` [tseries] - Input tseries object whose columns will be plotted as a stem graph.

Output arguments

- `H` [handle | numeric] - Handles to stems plotted.
- `Range` [numeric] - Actually plotted date range.

Options

See help on `tseries/plot` P352 and the built-in function `stem` for all options available.

Description

Example

---

subsasgn

Subscripted assignment for tseries objects

Syntax

```
X(Dates) = Values;
X(Dates,I,J,K,...) = Values;
```

Input arguments

- X [tseries] - Tseries object that will be assigned new observations.
- Dates [numeric] - Dates for which the new observations will be assigned.
- I, J, K, ... [numeric] - References to 2nd and higher dimensions of the tseries object.
- Values [numeric] - New observations that will assigned at specified dates.

Output arguments

- X [tseries] - Tseries object with newly assigned observations.

Description

Example

---

subsref

Subscripted reference function for tseries objects

Syntax returning numeric array

```
... = X(Dates)
... = X(Dates,...)
```

Syntax returning tseries object

```
... = X{Dates}
... = X{Dates,...}
```

Input arguments

- X [tseries] - Tseries object.
- Dates [numeric] - Dates for which the time series observations will be returned, either as a numeric array or as another tseries object.

Description

Example

---

trend

Estimate a time trend

Syntax

```
X = trend(X,range)
```

Input arguments

- X [tseries] - Input time series.
- Range [numeric | @all | char] - Range for which the trend will be computed; @all means the entire range of the input times series.

Output arguments

- X [tseries] - Output trend time series.

Options

- 'break=' [numeric | empty] - Vector of breaking points at which the trend may change its slope.
- 'connect=' [true | false] - Calculate the trend by connecting the first and the last observations.
- 'diff=' [true | false] - Estimate the trend on differenced data.
- 'log=' [true | false] - Logarithmise the input data, de-logarithmise the output data.
- 'season=' [true | false | 2 | 4 | 6 | 12] - Include deterministic seasonal factors in the trend.

Description

Example

---

tseries

Create new time series (tseries) object

Syntax

```
X = tseries()
X = tseries(Dates,Values)
X = tseries(Dates,Values,Comment)
```

Input arguments

- Dates [numeric | char] - Dates for which observations will be supplied; Dates do not need to be sorted in ascending order or create a continuous date range. If Dates is scalar and Values have multiple rows, then the date in Dates is interpreted as a startdate for the entire time series.
- Values [numeric | function_handle] - Numerical values (observations) arranged columnwise, or a function that will be used to create an N-by-1 array of values, where N is the number of Dates.
- Comment [char | cellstr] - Comment or comments attached to each column of observations.

Output arguments

- X [tseries] - New tseries object.

Description

Example

---

windex

Simple weighted or Divisia index

Syntax

```
Y = windex(X,W,Range)
```

Input arguments

- X [tseries] - Input times series.
- W [tseries | numeric] - Fixed or time-varying weights on the input time series.
- Range [numeric] - Range on which the Divisia index is computed.

Output arguments

- Y [tseries] - Weighted index based on X.

Options

- 'method=' ['divisia' | 'simple'] - Weighting method.
- 'log=' [true | false] - Logarithmise the input data before computing the index, delogarithmise the output data.

Description

Example

---

wmean

Weighted average of time series observations

Syntax

```
Y = wmean(X,RANGE,BETA)
```

Input arguments

- X [tseries] - Input tseries object whose data will be averaged column by column.
- RANGE [numeric] - Date range on which the weighted average will be computed.
- BETA [numeric] - Discount factor; the last observation gets a weight of of 1, the N-minus-1st observation gets a weight of BETA, the N-minus-2nd gets a weight of BETA^2, and so on.

Output arguments

- Y [numeric] - Array with weighted average of individual columns; the sizes of Y are identical to those of the input tseries object in 2nd and higher dimensions.

Description

Example

---

x12

Access to X13-ARIMA-SEATS seasonal adjustment program

Syntax with a single type of output requested

```
[Y,OutpFile,ErrFile,Model,X] = x12(X,...)
[Y,OutpFile,ErrFile,Model,X] = x12(X,Range,...)
```

Syntax with mutliple types of output requested

```
[Y1,Y2,...,OutpFile,ErrFile,Model,X] = x12(X,Range,...)
```

See the option 'output=' for the types of output data available from X12.

Input arguments

- X [tseries] - Input data that will seasonally adjusted or filtered by the Census X12 Arima; X must be a quarterly or monthly time series.
- Range [numeric | char | @all] - Date range on which the X12 will be run; @all means the entire on which the input time series is defined; Range may be omitted.

Output arguments

- Y, Y1, Y2, ... [tseries] - Requested output data, by default only one type of output is returned, the seasonlly adjusted data; see the option 'output='.
- OutpFile [cellstr] - Contents of the output log files produced by X12; each cell contains the log file for one type of output requested.
- ErrFile [cellstr] - Contents of the error files produced by X12; each cell contains the error file for one type of output requested.
- Model [struct] - Struct array with model specifications and parameter estimates for each of the ARIMA models fitted; Model matches the size of X is 2nd and higher dimensions.
- X [tseries] - Original input data with forecasts and/or backcasts appended if the options 'forecast=' and/or 'backcast=' are used.

Options

- 'backcast=' [numeric | 0] - 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 X13-ARIMA-SEATS manual. The backcast is included in the output argument X.
- 'cleanup=' [true | false] - Delete temporary X12 files when done; the temporary files are named iris_x12a.*.
- 'log=' [true | false] - Logarithmise the input data before, and de-logarithmise the output data back after, running x12.

- 'forecast=' [numeric | 0] - 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 X13-ARIMA-SEATS manual. The forecast is included in the output argument X.

- 'display=' [true | false] - Display X12 output messages in command window; if false the messages will be saved in a TXT file.

- 'dummy=' [tseries | empty] - User dummy variable or variables (in case of a multivariate tseries object) used in X13-ARIMA-SEATS regression; the dummy variables must also include values for forecasts and backcasts if you request them; the type of the dummy can be specified in the option 'dummyType='.

- 'dummyType=' ['ao' | 'holiday' | 'td'] - Type of the user dummy (which is specified through the option 'dummy='); the three basic types of dummies are additive outlier ('ao'), holiday flows ('holiday'), and trading days ('td'); see the X13-ARIMA-SEATS or X13-ARIMA documentation for more details (available from the U.S.Census Bureau website), look for the section on the REGRESSION spec, options 'user' and 'usertype'.

- 'mode=' ['auto' | 'add' | 'logadd' | 'mult' | 'pseudoadd' | 'sign'] - Seasonal adjustment mode (see help on the x11 specs in the X13-ARIMA-SEATS 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.

- 'maxIter=' [numeric | 1500] - Maximum number of iterations for the X12 estimation procedure. See help on the estimation specs in the X13-ARIMA-SEATS manual.

- 'maxOrder=' [numeric | [2,1]] - 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 X13-ARIMA-SEATS manual.

- 'missing=' [true | false] - 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.

- 'output=' [char | cellstr | 'SA'] - List of requested output data; the cellstr or comma-separated list can combine any number of the request specifications listed below in subsection Output request; See also help on the x11 specs in the X13-ARIMA-SEATS manual.

- 'saveAs=' [char | empty] - Name (or a whole path) under which X13-ARIMA-SEATS output files will be saved.

- 'specFile=' [char | 'default'] - Name of the X13-ARIMA-SEATS spec file; if 'default' the IRIS default spec file will be used, see description.

- 'tdays=' [true | false] - Correct for the number of trading days. See help on the x11regression specs in the X13-ARIMA-SEATS manual.

- 'tempDir=' [char | function_handle | '.'] - Directory in which X13-ARIMA-SEATS temporary files will be created; if the directory does not exist, it will be created at the beginning and deleted at the end of the execution (unless 'cleanup=' false).
- 'tolerance=' [numeric | 1e-5] - Convergence tolerance for the X13 estimation procedure. See help on the estimation specs in the X13-ARIMA-SEATS manual.

Description

Output requests

The option "output=' can combine any number of the following requests:

- 'SA' - seasonally adjusted series;
- 'SF' - seasonal factors;
- 'TC' - trend-cycle component;
- 'IR' - irregular component;
- 'MV' - the original input series with missing values fitted by running an estimated ARIMA model.

Missing observations

If you keep 'missing=' false (this is the default for backward compatibility), x12 will not run on series with in-sample missing observations, and a warning will be thrown.

If you set 'missing=' true, you allow for in-sample missing observations. The X13-ARIMA-SEATS program handles missing observations by filling in values predicted by the estimated ARIMA process. You can request the series with missing values filled in by including MV in the option 'output='.

Spec file

The default X13-ARIMA-SEATS spec file is +thirdparty/x12/default.spc. You can create your own spec file to include options that are not available through the IRIS interface. You can use the following pre-defined placeholders letting IRIS fill in some of the information needed (check out the default file):

- $series_data$ is replaced with a column vector of input observations;
- $series_freq$ is replaced with a number representing the date frequency: either 4 for quarterly, or 12 for monthly (other frequencies are currenlty not supported by X13-ARIMA-SEATS);

- $series_startyear$ is replaced with the start year of the input series;
- $series_startper$ is replaced with the start quarter or month of the input series;
- $transform_function$ is replaced with log or none depending on the mode selected by the user;
- $forecast_maxlead$ is replaced with the requested number of ARIMA forecast periods used to extend the series before seasonal adjustment.
- $forecast_maxlead$ is replaced with the requested number of ARIMA forecast periods used to extend the series before seasonal adjustment.
- $tolerance$ is replaced with the requested convergence tolerance in the estimation spec.
- $maxiter$ is replaced with the requested maximum number of iterations in the estimation spec.
- $maxorder$ is replaced with two numbers separated by a blank space: maximum order of regular ARIMA, and maximum order of seasonal ARIMA.
- $x11_mode$ is replaced with the requested mode: 'add' for additive, 'mult' for multiplicative, 'pseudoadd' for pseudo-additive, or 'logadd' for log-additive;
- $x12_save$ is replaced with the list of the requested output series: 'd10' for seasonals, 'd11' for final seasonally adjusted series, 'd12' for trend-cycle, 'd13' for irregular component.

Two of the placeholders, '$series_data$ and $x12_output$, are required; if they are not found in the spec file, IRIS throws an error.

Estimates of ARIMA model parameters

The ARIMA model specification, Model, is a struct with three fields:

- .spec - a cell array with the first cell giving the structure of the non-seasonal ARIMA, and the second cell giving the structure of the seasonal ARIMA; both specifications follow the usual Box-Jenkins notation, e.g. [0 1 1].
- .ar - a numeric array with the point estimates of the AR coefficients (non-seasonal and seasonal).
- .ma - a numeric array with the point estimates of the MA coefficients (non-seasonal and seasonal).

Example

Run X12 on the entire range of a time series:

```
xsa = x12(x);
xsa = x12(x,Inf);
xsa = x12(x,@all);
xsa = x12(x,get(x,'range'));
```

yearly

Display tseries object one calendar year per row

Syntax

```
yearly(X)
```

Input arguments

- X [tseries] - Tseries object that will be displayed one full year of observations per row.

Description

The functon yearly currently works for tseries with monthly, bi-monthly, quarterly, and half-yearly frequency only.

Example

Create a quarterly tseries, and use yearly to display it one calendar year per row.

```
>> x = tseries(qq(2000,3):qq(2002,2),@rand)
x =
    tseries object: 8-by-1
    2000Q3:  0.95537
    2000Q4:  0.68029
    2001Q1:  0.86056
    2001Q2:  0.93909
    2001Q3:  0.68019
    2001Q4:  0.91742
    2002Q1:  0.25669
    2002Q2:  0.88562
    ''
    user data: empty
>> yearly(x)
    tseries object: 8-by-1
    2000Q1-2000Q4:        NaN           NaN     0.9553698     0.6802907
```

```
2001Q1-2001Q4:  0.8605621     0.9390935      0.680194     0.9174237
2002Q1-2002Q4:  0.2566917     0.8856181           NaN           NaN
''
user data: empty
```

19 Time-Recursive Expressions (trec Objects)

Time-recursive subscript objects (trec objects) allow creating and evaluating time-recursive expressions based on tseries P306 objects. Read below carefully when IRIS fails to evaluate time-recursive expessions correctly.

Trec methods:

Constructor

- trec P388 - Create new recursive time subscript object.

Creating lags and leads

- plus P387 - Create time-recursive lead of tseries object.
- minus P387 - Create time-recursive lag of tseries object.

Using Time-Recursive Subscripts

Time-recursive expressions are expressions that are evaluated period by period, with each result assigned immediately to the left-hand side tseries variable, and used in subsequent periods evaluated afterwards.

To construct and evaluate time-recursive expressions, use tseries referenced by a trec object, or a lag or lead created from a trec object. Every tseries object on both the left-hand side (i.e. the variable under construction) and the right-hand side (i.e. the variables in the expression that is being evaluated) must be referenced by a trec object (or possibly a lag or lead). When referencing a tseries object by a trec, you can use either curly braces, `{...}`, or round brackets, `(...)`; there is no difference between them in time-recursive expressions.

★★★ See the description below of situations when IRIS fails to evaluate time-recursive expressions correctly, and how to avoid/fix such situations.

Example

Construct an autoregressive sequence starting from an initial value of 10 with a autoregressive coefficient 0.8 between periods 2010Q1 and 2020Q4:

```
T = trec(qq(2010,1):qq(2020,4));
x = tseries(qq(2009,4),10);
x{T} = 0.8*x{T-1};
```

Example

Construct a first-order autoregressive process, x, with normally distributed innovations, e:

```
T = trec(qq(2010,1):qq(2020,4));
x = tseries(qq(2009,4),10);
e = tseries(qq(2010,1):qq(2020,4),@randn);
x{T} = (1-0.8)*10 + 0.8*x{T-1} + e{T};
```

Example

Construct a second-order log-autoregressive process going backward from year 2020 to year 2000.

```
T = trec(yy(2020):-1:yy(2000));
b = tseries();
b(yy(2022)) = 1.56;
b(yy(2021)) = 1.32;
b{T} = b{T+1}^1.2 / b{T+2}^0.6;
```

Example

Construct the first 20 numbers of the Fibonacci sequence:

```
T = trec(3:20);
f = tseries(1:2,1);
f{T} = f{T-1} + f{T-2};
```

When IRIS Fails to Evaluate Time-Recursive Expressions Correctly

★★★ IRIS fails to evaluate time-recursive expressions correctly (without any indication of an error) when the following two circumstances occur at the same time:

- At least one tseries object on the right-hand side has been created by copying the left-hand side tseries object with no further manipulation.
- The time series used in the expression are within a database (struct), or a cell array;

Under these circumstances, the right-hand side tseries variable will be assigned (updated with) the results calculated in iteration as if it were the tseries variable on the left-hand side.

Example

Create a database with two tseries. Create one of the tseries by simply copying the other (i.e. plain assignment with no further manipulation).

```
d = struct();
d.x = tseries(1:10,1);
d.y = d.x;

T = trec(2:10);
d.x{T} = 0.8*d.y{T-1}; % Fails to evaluate correctly.
```

The above time-recursive expression will be incorrectly evaluated as if it were `d.x{T} = 0.8*d.x{T-1}`. However, when the tseries objects are not stored within a database (struct) but exist as stand-alone variables, the expression will evaluate correctly:

```
x = tseries(1:10,1);
y = x;

T = trec(2:10);
x{T} = 0.8*y{T-1}; % Evaluates correctly.
```

Workaround when Time-Recursive Expressions Fail

★★★ To evaluate the expression correctly, simply apply any kind of operator or function to the tseries `d.y` before it enters the time-recursive expression. Below are examples of some simple manipulations that do the job without changing the tseries `d.y`:

```
d = struct();
d.x = tseries(1:10,1);
d.y = 1*d.x;
```

or

```
d = struct();
d.x = tseries(1:10,1);
d.y = d.x{:};
```

or

```
d = struct();
d.x = tseries(1:10,1);
d.y = d.x;
d.y = d.y + 0;
```

Reference page for trec

---

minus

Create time-recursive lag of tseries object

Syntax

```
X{T-K}
```

Input arguments

- X [tseries] - Tseries object whose time-recursive lag will be created.
- T [trec] - Initialized trec object.
- K [numeric] - Integer scalar specifying the lag.

Description

The tseries object, X, referenced by T-K in a time-recursive expression will, in each iteration, return a value that corresponds to period t-K, where t is the currently processed date from the vector of dates (or date range) associated with the trec object, T.

---

plus

Create time-recursive lead of tseries object

Syntax

```
X{T+K}
```

Input arguments

- X [tseries] - Tseries object whose time-recursive lead will be created.
- T [trec] - Initialized trec object.
- K [numeric] - Integer scalar specifying the lead.

Description

The tseries object, X, referenced by T+K in a time-recursive expression will, in each iteration, return a value that corresponds to period t+K, where t is the currently processed date from the vector of dates (or date range) associated with the trec object, T.

---

trec

Create new recursive time subscript object

Syntax

```
T = trec(Dates)
```

Input arguments

- Dates [numeric] - Vector of dates or date range on which the final time-recursive expression will be evaluated.

Output arguments

- T [trec] - New time-recursive subscript object.

Description

Time-recursive subscript objects are used to reference tseries objects on both the left-hand side and the right-hand side of a time-recursive assignment. The assignment is then evaluated for each date in Dates, from the first to the last.

See more on time-recursive expressions in Contents P384, including the description of instances in which IRIS fails to evaluate the time-recursive expressions correctly.

Example

Construct a first-order autoregressive process with normally distributed residuals:

```
T = trec(qq(2010,1):qq(2020,4));
x = tseries(qq(2009,4),10);
e = tseries(qq(2010,1):qq(2020,4),@randn);
x(T) = 10 + 0.8*x(T-1) + e(T);
```

20 Basic Database Management

Loading and saving databases

- dbload P400 - Create database by loading CSV file.
- dbsave P415 - Save database as CSV file.
- xls2csv P423 - Convert XLS file to CSV file.

Getting information about databases

- dbnames P405 - List of database entries filtered by name and/or class.
- dbprintuserdata P413 - Print names of database tseries along with specified fields of their userdata.
- dbrange P413 - Find a range that encompasses the ranges of the listed tseries objects.
- dbsearchuserdata P418 - Search database to find tseries by matching the content of their userdata fields.
- dbuserdatalov P420 - List of values found in a specified user data field in tseries objects.

Converting databases

- array2db P391 - Convert numeric array to database.
- db2array P392 - Convert tseries database entries to numeric array.
- db2tseries P393 - Combine tseries database entries in one multivariate tseries object.

Batch processing

- dbbatch P393 - Run a batch job to create new database fields.
- dbclip P396 - Clip all tseries entries in database down to specified date range.
- dbcol P397 - Retrieve the specified column or columns from database entries.
- dbcomment P398 - Create model-based comments for database tseries entries.
- dbfun P398 - Apply function to database fields.
- dbplot P408 - Plot from database.
- dbpage P407 - Retrieve the specified page or pages from database entries.
- dbredate P414 - Redate all tseries objects in a database.

Combining and splitting databases

- dboverlay P406 - Combine tseries observations from two or more databases.
- dbmerge P403 - Merge two or more databases.
- dbminuscontrol P404 - Create simulation-minus-control database.
- dbsplit P419 - Split database into mutliple databases.

Overloaded operators for databases

- - P421 - Remove entries from a database.
- * P422 - Keep only the database entries that are on the list.
- + P422 - Merge entries from two databases together.

Getting on-line help on database functions

```
help dbase
help dbase/function_name
```

---

array2db

Convert numeric array to database

Syntax

```
D = array2db(X,Range,List)
```

Input arguments

- X [numeric] - Numeric array with individual time series in columns.
- Dates [numeric] - Vector of dates for individual rows of X.
- List [cellstr | char] - List of names for time series in individual columns of X.

Output arguments

- D [struct] - Output database.

Description

Example

---

db2array

Convert tseries database entries to numeric array

Syntax

```
[X,Incl,Range] = db2array(D)
[X,Incl,Range] = db2array(D,List)
[X,Incl,Range] = db2array(D,List,Range,...)
```

Input arguments

- `D` [struct] - Input database with tseries objects that will be converted to a numeric array.
- `List` [char | cellstr] - List of tseries names that will be converted to a numeric array; if not specified, all tseries entries found in the input database, `D`, will be included in the output arrays, `X`.
- `Range` [numeric | `Inf`] - Date range; `Inf` means a range from the very first non-NaN observation to the very last non-NaN observation.

Output arguments

- `X` [numeric] - Numeric array with observations from individual tseries objects in columns.
- `Incl` [cellstr] - List of tseries names that have been actually found in the database.
- `Range` [numeric] - Date range actually used; this output argument is useful when the input argument `Range` is missing or `Inf`.

Description

The output array, `X`, is always NPer-by-NList-by-NAlt, where NPer is the length of the `Range` (the number of periods), NList is the number of tseries included in the `List`, and NAlt is the maximum number of columns that any of the tseries included in the `List` have.

If all tseries data have the same size in 2nd and higher dimensions, the output array will respect that size in 3rd and higher dimensions. For instance, if all tseries data are NPer-by-2-by-5, the output array will be NPer-by-Nx-by-2-by-5. If some tseries data have unmatching size in 2nd or higher dimensions, the output array will be always a 3D array with all higher dimensions unfolded in 3rd dimension.

If some tseries data have smaller size in 2nd or higher dimensions than other tseries entries, the last available column will be repeated for the missing columns.

Example

---

db2tseries

Combine tseries database entries in one multivariate tseries object

Syntax

```
[X,Incl,Range] = db2tseries(D,List,Range)
```

Input arguments

- `D` [struct] - Input database with tseries objects that will be combined in one multivariate tseries object.
- `List` [char | cellstr] - List of tseries names that will be combined.
- `Range` [numeric | Inf] - Date range.

Output arguments

- `X` [numeric] - Combined multivariate tseries object.
- `Incl` [cellstr] - List of tseries names that have been actually found in the database.
- `Range` [numeric] - The date range actually used.

---

dbbatch

Run a batch job to create new database fields

Syntax

```
[D,Processed,Added] = dbbatch(D,NewName,Expr,...)
```

Input arguments

- D [struct] - Input database.
- NewName [char] - Pattern that will be used to create names for new database fields based on the existing ones; use '$0' to refer to the name of the currently processed database field; use '$1', '$2', etc. to refer to tokens captured in regular expression specified in the 'namefilter=' option.
- Expr [char] - Expression that will be evaluated on a selection of existing database entries to create new database entries; the expression can include '$0', '$1', etc.

Output arguments

- D [struct] - Output database.
- Processed [cellstr] - List of database fields that have been used to create new fields.
- Added [cellstr] - List of new database fields created by evaluating Expr on the corresponding field in Processed.

Options

- 'classFilter=' [char | Inf] - From the existing database entries, select only those that are objects of the specified class or classes, and evaluate the expression Expr on these.
- 'fresh=' [true | false] - If true, the output database will only contain the newly created entries; if false the output database will also include all the entries from the input database.
- 'nameFilter=' [char | empty] - From the existing database entries, select only those that match this regular expression, and evaluate the expression Expr on these.
- 'nameList=' [cellstr | Inf] - Evaluate the COMMAND on this list of existing database entries.
- 'stringList=' [cellstr | empty] - Evaluate the expression Expr on this list of strings; the strings do not need to be names existing in the database; this options can be comined with 'nameFilter=', 'nameList=', and/or 'classFilter=' to narrow the selection.

Description

This function is primarily meant to create new database fields, each based on an existing one. If you, on the otherhand, only wish to modify a number of existing fields without adding any new ones, use dbfun P398 instead.

The expression Expr is evaluated in the caller workspace, and hence may refer to any variables existing in the workspace, not only to the database and its fields.

To convert the strings $0, $1, $2, etc. to lower case or upper case, use the dot or colon syntax: $.0, $.1, $.2 for ower case, and $:0, $:1, $:2 for upper case.

Failure

The function dbbatch will always fail when called on a sub-database from within a function (as opposed to a script). A sub-database is a struct within a struct, a struct within a cell array, a struct within an array of structs, etc.

```
function ...
    d.e = dbbatch(d.e,...);
    ...
end

function ...
    d{1} = dbbatch(d{1},...);
    ...
end

function ...
    d(1) = dbbatch(d(1),...);
    ...
end
```

Example

For each field (all assumed to be tseries) create a first difference, and name the new series DX where X is the name of the original series.

```
d = dbbatch(d,'D$0','diff(d.$0)');
```

Note that the original series will be presered in the database, together with the newly created ones.

Example

Suppose that in database D you want to seasonally adjust all time series whose names end with _u, and give these seasonally adjusted series names without the _u.

```
d = dbbatch(d,'$1','x12(d.$0)','nameFilter','(.*)u');
```

or, if you want to make sure only tseries objects will be selected (in case there are database entries ending with a u other than tseries objects)

```
d = dbbatch(d,'$1','x12(d.$0)', ...
    'nameFilter=','(.*)u','classFilter=','tseries');
```

---

dbclip

Clip all tseries entries in database down to specified date range

Syntax

```
D = dbclip(D,Range)
```

Input arguments

- `D` [struct] - Database or nested databases with tseries objects.
- `Range` [numeric | cell] - Range or a cell array of ranges to which all tseries objects will be clipped; multiple ranges can be specified, each for a different date frequency/periodicity.

Output arguments

- `D` [struct] - Database with tseries objects cut down to `range`.

Description

This functions looks up all tseries objects within the database `d`, including tseries objects nested in sub-databases, and cuts off any values preceding the start date of `Range` or following the end date of `range`. The tseries object comments, if any, are preserved in the new database.

If a tseries entry does not match the date frequency of the input range, a warning is thrown.

Multiple ranges can be specified in `Range` (as a cell array), each for a different date frequency/periodicity (i.e. one or more of the following: monthly, bi-monthly, quarterly, half-yearly, yearly, indeterminate). Each tseries entry will be clipped to the range that matches its date frequency.

Example

```
d = struct();
d.x = tseries(qq(2005,1):qq(2010,4),@rand);
d.y = tseries(qq(2005,1):qq(2010,4),@rand)

d =
   x: [24x1 tseries]
   y: [24x1 tseries]

dbclip(d,qq(2007,1):qq(2007,4))

ans =
    x: [4x1 tseries]
    y: [4x1 tseries]
```

---

dbcol

Retrieve the specified column or columns from database entries

Syntax

```
D = dbpage(D,K)
```

Input arguments

- `D` [struct] - Input database with (possibly) multivariate tseries objects and numeric arrays.
- `K` [numeric | logical | `'end'`] - Column or columns that will be retrieved from each tseries object or numeric array in in the input database, `D`, and returned in the output database.

Output arguments

- `D` [struct] - Output database with tseries objects and numeric arrays reduced to the specified column.

Description

Example

---

dbcomment

Create model-based comments for database tseries entries

Syntax

```
D = dbcomment(D,M)
```

Input arguments

- `D` [struct] - Database.
- `M` [model] - Model object.

Output arguments

- `D` [struct] - Database where every tseries entry is (if possible) assigned a comment based on the description of a model variable or parameter found in the model object, `M`.

Description

Example

---

dbfun

Apply function to database fields

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[D,Flag,ErrList,WarnList] = dbfun(Func,D,...)
[D,Flag,ErrList,WarnList] = dbfun(Func,D,~D2,~D3,...,~Dk,...)
```

Input arguments

- `Func` [function_handle | char] - Function that will be applied to each field.
- `D` [struct] - Primary input database whose fields will be processed by the function `Func`.
- `~D2`, `~D3`, `...` [struct] - Secondary input databases whose fields will be passed into `Func` (`Func` accepts more than one input argument).

Output arguments

- `D` [struct] - Output database whose fields will be created by applying `Func` to each field of the input database or databases.
- `Flag` [`true` | `false`] - True if no error occurs when evaluating the function.
- `ErrList` [cellstr] - List of fields on which the function has thrown an error.
- `WarnList` [cellstr] - List of fields on which the function has thrown a warning.

Options

- `'subdbase='` [`true` | `false`] - Go through all sub-databases (i.e. struct fields within the input struct), applying the function `Func` to their fields, too.
- `'classFilter='` [cell | cellstr | rexp | `@all`] - Apply `Func` only to database fields whose class is on the list or matches the regular expression; `@all` means all fields in the input database `D` will be processed.
- `'fresh='` [`true` | `false`] - Remove unprocessed entries from the output database.
- `'nameList='` [cell | cellstr | rexp | `@all`] - Apply `Func` only to this list of database field names or names that match this regular expression; `@all` means all entries in the input database `D` wil be processed.
- `'ifError='` [`'NaN'` | `'remove'`] - What to do with the database entry if an error occurs when the entry is being evaluated.
- `'ifWarning='` [`'keep'` | `'NaN'` | `'remove'`] - What to do with the database entry if an error occurs when the entry is being evaluated.

Description

Example

```
d = struct( );
d.a = [1, 2];
d.b = tseries(1:3,@ones);
d = dbfun( @(x) 2*x, d)

d =
    a: [2 4]
    b: [3x1 tseries]

d.b
ans =
    tseries object: 3-by-1
    1:  2
    2:  2
    3:  2
    ''
    user data: empty
    export files: [0]
```

---

dbload

Create database by loading CSV file

Syntax

```
D = dbload(FName, ...)
D = dbload(D,FName, ...)
```

Input arguments

- `FName` [char | cellstr] - Name of the Input CSV data file or a cell array of CSV file names that will be combined.
- `D` [struct] - An existing database (struct) to which the new entries from the input CSV data file entries will be added.

Output arguments

- D [struct] - Database created from the input CSV file(s).

Options

- 'case=' ['lower' | 'upper' | empty] - Change case of variable names.
- 'commentRow=' [char | cellstr | {'comment','comments'}] - Label at the start of row that will be used to create tseries object comments.
- 'dateFormat=' [char | 'YYYYFP'] - Format of dates in first column.
- 'delimiter=' [char | ','] - Delimiter separating the individual values (cells) in the CSV file; if different from a comma, all occurences of the delimiter will replaced with commas – note that this will also affect text in comments.
- 'firstDateOnly=' [true | false] - Read and parse only the first date string, and fill in the remaining dates assuming a range of consecutive dates.
- 'freq=' [0 | 1 | 2 | 4 | 6 | 12 | 365 | 'daily' | empty] - Advise frequency of dates; if empty, frequency will be automatically recognised.
- 'freqLetters=' [char | 'YHQBM'] - Letters representing frequency of dates in date column.
- 'inputFormat=' ['auto' | 'csv' | 'xls'] - Format of input data file; 'auto' means the format will be determined by the file extension.
- 'nameRow=' [char | numeric | {'','Variables'}] - String, or cell array of possible strings, that is found at the beginning (in the first cell) of the row with variable names, or the line number at which the row with variable names appears (first row is numbered 1).
- 'nameFunc=' [cell | function_handle | empty] - Function used to change or transform the variable names. If a cell array of function handles, each function will be applied in the given order.
- 'nan=' [char | NaN] - String representing missing observations (case insensitive).
- 'preProcess=' [function_handle | cell | empty] - Apply this function, or cell array of functions, to the raw text file before parsing the data.
- 'select=' [char | cellstr | empty] - Only database entries included on this list will be read in and returned in the output database D; entries not on this list will be discarded.
- 'skipRows=' [char | cellstr | numeric | empty] - Skip rows whose first cell matches the string or strings (regular expressions); or, skip a vector of row numbers.

- 'userData=' [char | Inf] - Field name under which the database userdata loaded from the CSV file (if they exist) will be stored in the output database; Inf means the field name will be read from the CSV file (and will be thus identical to the originally saved database).
- 'userDataField=' [char | '.'] - A leading character denoting userdata fields for individual time series; if empty, no userdata fields will be read in and created.
- 'userDataFieldList=' [cellstr | numeric | empty] - List of row headers, or vector of row numbers, that will be included as user data in each time series.

Description

Use the 'freq=' option whenever there is ambiguity in intepreting the date strings, and IRIS is not able to determine the frequency correctly (see Example).

Structure of CSV database files

The minimalist structure of a CSV database file has a leading row with variables names, a leading column with dates in the basic IRIS format, and individual columns with numeric data:

```
+---------+---------+---------+--
|         |       Y |       P |
+---------+---------+---------+--
|  2010Q1 |       1 |      10 |
+---------+---------+---------+--
|  2010Q2 |       2 |      20 |
+---------+---------+---------+--
|         |         |         |
```

You can add a comment row (must be placed before the data part, and start with a label 'Comment' in the first cell) that will also be read in and assigned as comments to the individual tseries objects created in the output database.

```
+---------+---------+---------+--
|         |       Y |       P |
+---------+---------+---------+--
| Comment |  Output |  Prices |
+---------+---------+---------+--
|  2010Q1 |       1 |      10 |
+---------+---------+---------+--
|  2010Q2 |       2 |      20 |
+---------+---------+---------+--
|         |         |         |
```

You can use a different label in the first cell to denote a comment row; in that case you need to set the option 'commentRow=' accordingly.

All CSV rows whose names start with a character specified in the option 'userdataField=' (a dot by default) will be added to output tseries objects as fields of their userdata.

```
+---------+---------+---------+--
|         |       Y |       P |
+---------+---------+---------+--
| Comment |  Output |  Prices |
+---------+---------+---------+--
| .Source |   Stat  |  IMFIFS |
+---------+---------+---------+--
| .Update | 17Feb11 | 01Feb11 |
+---------+---------+---------+--
| .Units  | Bil USD |  2010=1 |
+---------+---------+---------+--
|  2010Q1 |       1 |      10 |
+---------+---------+---------+--
|  2010Q2 |       2 |      20 |
+---------+---------+---------+--
|         |         |         |
```

Example

Typical example of using the 'freq=' option is a quarterly database with dates represented by the corresponding months, such as a sequence 2000-01-01, 2000-04-01, 2000-07-01, 2000-10-01, etc. In this case, you can use the following options:

```
d = dbload('filename.csv','dateFormat','YYYY-MM-01','freq',4);
```

---

dbmerge

Merge two or more databases

Syntax

```
D = dbmerge(D1,D2,...)
```

Input arguments

- D1, D2, ... [struct] - Input databases whose entries will be combined in the output datase.

Output arguments

- D [struct] - Output database that combines entries from all input database; if some entries are found in more than one input databases, the last occurence is used.

Description

Example

```
d1 = struct('a',1,'b',2);
d2 = struct('a',10,'c',20);
d = dbmerge(d1,d2)
d =
   a: 10
   b: 2
   c: 20
```

---

dbminuscontrol

Create simulation-minus-control database

Syntax

```
[D,C] = dbminuscontrol(M,D)
[D,C] = dbminuscontrol(M,D,C)
```

Input arguments

- M [model] - Model object on which the databases D and C are based.
- D [struct] - Simulation database.
- C [struct] - Control database; if the input argument C is omitted the steady-state database of the model M is used for the control database.

Output arguments

- `D` [struct] - Simulation-minus-control database, in which all log variables are `d.x/c.x`, and all other variables are `d.x-c.x`.
- `C` [struct] - Control database.

Options

- `'fresh='` [`true` | `false`] - If `true`, the output database will only contain entries corresponding to model variables in `M`; if `false` all other entries found in the input database will be also kept in the output database.

Description

Example

We run a shock simulation in full levels using a steady-state (or balanced-growth-path) database as input, and then compute the deviations from the steady state.

```
d = sstatedb(m,1:40);
... % Set up a shock or shocks here.
s = simulate(m,d,1:40);
s = dboverlay(d,s);
s = dbminuscontrol(m,s,d);
```

The above block of code is equivalent to this one:

```
d = zerodb(m,1:40);
... % Set up a shock or shocks here.
s = simulate(m,d,1:40,'deviation=',true);
s = dboverlay(d,s);
```

---

dbnames

List of database entries filtered by name and/or class

Syntax

```
List = dbnames(D,...)
```

Input arguments

- D [struct] - Input database.

Output arguments

- List [cellstr] - List of input database entries that pass the name or class test.

Options

- 'nameFilter=' [cellstr | char | rexp | @all] - List of names or regular expression against which the database entry names will be matched; @all means all names will be matched.
- 'classFilter=' [cellstr | char | rexp | @all] - List of names or regular expression against which the database entry class names will be matched; @all means all classes will be matched.

Description

Example

Notice the differences in the following calls to dbnames:

```
dbnames(d,'nameFilter=','L_')
```

matches all names that contain 'L_' (at the beginning, in the middle, or at the end of the string), such as 'L_A', 'DL_A', 'XL_', or just 'L_'.

```
dbnames(d,'nameFilter=','^L_')
```

matches all names that start with 'L_', such as 'L_A' or 'L_', but not 'DL_A'. Finally,

```
dbnames(d,'nameFilter=','^L_.')
```

matches all names that start with 'L_' and have at least one more character after that, such as 'L_A' but not 'L_' or 'L_RX'.

---

dboverlay

Combine tseries observations from two or more databases

Syntax

```
D = dboverlay(D,D1,D2,...)
```

Input arguments

- `D` [struct] - Primary input database.
- `D1`, `D2`, ... [struct] - Databases whose tseries observations will be used to extend or overwrite observations in the tseries objects of the same name in the primary database.

Output arguments

- `D` [struct] - Output database.

Description

If more than two databases are combined then they are processed one-by-one: the first is combined with the second, then the result is combined with the third, and so on, using the following rules:

- If two non-empty tseries objects with the same frequency are combined, the observations are spliced together. If some of the observations overlap the observations from the second tseries are used.
- If two empty tseries objects are combined the first is used.
- If a non-empty tseries is combined with an empty tseries, the non-empty one is used.
- If two objects are combined of which at least one is a non-tseries object, the second input object is used.

Example

---

dbpage

Retrieve the specified page or pages from database entries

Syntax

```
D = dbpage(D,K)
```

Input arguments

- D [struct] - Input database with (possibly) multivariate tseries objects and numeric arrays.
- K [numeric | logical | 'end'] - Page or pages that will be retrieved from each tseries object or numeric array in in the input database, D, and returned in the output database.

Output arguments

- D [struct] - Output database with tseries objects and numeric arrays reduced to the specified page.

Description

Example

---

dbplot

Plot from database

Syntax

```
[FF,AA,PDb] = dbplot(D,List,Range,...)
[FF,AA,PDb] = dbplot(D,Range,List,...)
[FF,AA,PDb] = dbplot(D,List,...)
[FF,AA,PDb] = dbplot(D,Range,...)
[FF,AA,PDb] = dbplot(D,...)
```

Input arguments

- D [struct] - Database with input data.
- List [cellstr | rexp] - List of expressions (or labelled expressions) that will be evaluated and plotted in separate graphs; if not specified, all time series name found in the input database D will be plotted. Alternatively, List can be a regular expression (rexp object), which will be matched against all time series names in the input database.
- Range [numeric] - Date range; if not specified, the function dbrange P413 will be used to determined the plotted range (same for all graphs).

Output arguments

- FF [numeric] - Handles to figures created by qplot.
- AA [cell] - Handles to axes created by qplot.
- PDB [struct] - Database with actually plotted series.

Options

- 'addClick=' [true | false] - Make axes expand in a new graphics figure upon mouse click.
- 'captions=' [cellstr | @comment | *empty*] - Strings that will be used for titles in the graphs that have no title in the q-file.
- 'clear=' [numeric | empty] - Serial numbers of graphs (axes objects) that will not be displayed.
- 'dbSave=' [cellstr | empty] - Options passed to dbsave when 'saveAs=' is used.
- 'deviationsFrom=' [numeric | empty] - Each expression in List that starts with a @ or # (see Description) will be reported in deviations from this specified date.
- 'deviationsTimes=' [numeric | empty] - Used only if 'deviationsFrom=' is non-empty; each expression in List that starts with a @ or # (see Description) will be reported in deviations multiplied by this number.
- 'drawNow=' [true | false] - Call Matlab drawnow function upon completion of all figures.
- 'grid=' [true | false] - Add grid lines to all graphs.
- 'highlight=' [numeric | cell | empty] - Date range or ranges that will be highlighted.
- 'interpreter=' ['latex' | 'none'] - Interpreter used in graph titles.
- 'mark=' [cellstr | empty] - Marks that will be added to each legend entry to distinguish individual columns of multivariated tseries objects plotted.
- 'maxPerFigure=' [numeric | 36] - Maximum number of graphs in one figure window; if the actual graph count exceeds maxPerFigure, the option 'subplot=' is adjusted automatically, and new figure windows are opened as needed.
- 'overflow=' [true | false] - Open automatically a new figure window if the number of subplots exceeds the available total; 'overflow='false means an error will occur instead.
- 'plotFunc=' [@bar | @hist | *@plot* | @plotcmp | @plotpred | @stem | cell] - Plot function used to create the graphs; use a cell array, {plotFunc,...} to specify extra input arguments that will be passed into the plotting function.

- 'prefix=' [char | 'P%g_'] - Prefix (a sprintf format string) that will be used to precede the name of each entry in the PDb database.
- 'round=' [numeric | Inf] - Round the input data to this number of decimals before plotting.
- 'saveAs=' [char | empty] - File name under which the plotted data will be saved either in a CSV data file or a PS graphics file; you can use the 'dbsave=' option to control the options used when saving CSV.
- 'style=' [struct | empty] - Style structure that will be applied to all figures and their children created by the qplot function.
- 'subplot=' ['auto' | numeric] - Default subplot division of figures, can be modified in the q-file.
- 'sstate=' [struct | model | empty] - Database or model object from which the steady-state values referenced to in the quick-report file will be taken.
- 'style=' [struct | empty] - Style structure that will be applied to all created figures upon completion.
- 'transform=' [function_handle | empty] - Function that will be used to trans
- 'tight=' [true | false] - Make the y-axis in each graph tight.
- 'vLine=' [numeric | empty] - Dates at which vertical lines will be plotted.
- 'zeroLine=' [true | false] - Add a horizontal zero line to graphs whose y-axis includes zero.

Description

The function dbplot opens a new figure window (as many as needed to accommodate all graphs given the option 'subplot='), and creates a graph for each entry in the cell array List.

List can contain either the names of database fields, or expressions referring to database fields; these expressions will be then evaluated in the input database context. You can also add labels (that will be displayed as graph titles) enclosed in double quotes and preceding the expressions. Alternatively, you can specify titles through the option 'captions='. At the beginning of the expression, you can use one of the following marks:

- ^ (a hat symbol) means the function specified in the option 'transform=' will not be applied to that expression;
- @ (an at symbol) in combination with the option 'deviationFrom=' means that the deviations will reported in multiplicative form (i.e. the actual value divided by the base period value).
- # (a hash symbol) in combination with the option 'deviationFrom=' means that the deviations will reported in additive form (i.e. the actual value minus the base period value).

Example

The following command will plot the time series x and y as deviations from 1 multiplied by 100 (see the option 'transform='), and the time series z as it is (because of the ^ symbol at the beginning). The first series will be labeled simply 'x', while the last two series will be labeled 'Series y' and 'Series z', respectively.

```
dbplot(d,qq(2010,1):qq(2015,4), ...
   { 'x', '"Series y" y', '^"Series z"' }, ...
   'transform=',@(x) 100*(x-1));
```

Example

The following command will plot the time series x and y as deviations from year 2000; x will be computed as additive deviations (i.e. the base period value will be subtracted from its observations) whereas y will be computed as a multiplicative deviations (i.e. the observations will be divided by the base period value). The last time series z will not be transforme.d

```
dbplot(d,yy(2000):yy(2010), ...
   { '# x', '@ y', 'z' }, ...
   'deviationsFrom=',yy(2000));
```

Example

The following command will plot all time series found in the database that start with 'a'.

```
dbplot(d,rexp('^a.*'));
```

Example

Create an example database with the following fields: c, ctrend, y, ytrend, k, ktrend (the exact way these series are created is, of course, irrelevant):

```
range = qq(2000,1):qq(2004,4);
s = struct();
s.c = 1+cumsum( tseries(range,@rand)/10 );
s.ctrend = hpf(s.c);
s.y = 1+cumsum( tseries(range,@rand)/10 );
```

```
s.ytrend = hpf(s.y);
s.k = 1+ cumsum( tseries(range,@rand)/10 );
s.ktrend = hpf(s.k);
disp(s);
```

Plot the individual series against their respective trends, each in its own graph:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' } );
```

To automate this task, create the list of expressions to be plotted using the standard Matlab function `strcat`:

```
list = {'c','y','k'};
plotList = strcat( '[' , list , ',' , list , 'trend]' );
disp(plotList);
dbplot(s,range,plotList);
```

In the case of some complex transformation(s), e.g.

```
dbplot(s,range, { ...
    '100*log([c,ctrend])',  ...
    '100*log([y,ytrend])', ...
    '100*log([k,ktrend])' } );
```

use the option `'transform='` to apply the specified function to all series before they get plotted:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' }, ...
    'transform=',@(x) 100*log(x) );
```

If some graphs need to be excluded from `'transform='`, use a hat ^ at the beginning of the expression:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '^[k,ktrend]' }, ...
    'transform=',@(x) 100*log(x) );
```

Include titles for the individual graphs in double quotes at the beginning of each expression:

```
dbplot(s,range, { ...
    '"Consumption" [c,ctrend]', ...
    '"Output" [y,ytrend]', ...
    '"Capital" [k,ktrend]' } );
```

or alternatively use the option 'captions=' to do the same thing:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' }, ....
    'captions=',{'Consumption','Output','Capital'} );
```

---

dbprintuserdata

Print names of database tseries along with specified fields of their userdata

Syntax

```
dbprintuserdata(D,Fields,...)
```

Input arguments

- D [struct] - Database whose tseries objects will be reported.
- Fields [char | cellstr] - Names of the userdata fields whose content will printed (if char or numeric scalar).

Options

- 'output=' ['html' | 'prompt'] - Where to display the information.

Description

Example

---

dbrange

Find a range that encompasses the ranges of the listed tseries objects

Syntax

Input arguments marked with a ~ (tilde) sign may be omitted.

```
[Range,FreqList] = dbrange(D,~List,...)
```

Input arguments

- D [struct] - Input database.
- ~List [cellstr | rexp | @all] - List of time series that will be included in the range search or a regular expression that will be matched to compose the list; @all means all tseries objects existing in the input databases will be included; may be omitted.

Output arguments

- Range [numeric | cell] - Range that encompasses the observations of the tseries objects in the input database; if tseries objects with different frequencies exist, the ranges are returned in a cell array.
- FreqList [numeric] - Vector of date frequencies coresponding to the returned ranges.

Options

- 'startDate=' ['maxRange' | 'minRange'] - 'maxRange' means the output Range will start at the earliest start date among all time series included in the search; 'minRange' means the range will start at the latest start date.
- 'endDate=' ['maxRange' | 'minRange'] - 'maxRange' means the range will end at the latest end date among all time series included in the search; 'minRange' means the range will end at the earliest end date.

Description

Example

---

dbredate

Redate all tseries objects in a database

Syntax

```
D = redate(D,OldDate,NewDate)
```

Input arguments

- `D` [struct] - Input database with tseries objects.
- `OldDate` [numeric] - Base date that will be converted to a new date in all tseries objects.
- `NewDate` [numeric] - A new date to which the base date `OldDate` will be changed in all tseries objects; `newDate` need not be the same frequency as `OldDate`.

Output arguments

- `d` [struct] - Output database where all tseries objects have identical data as in the input database, but with their time dimension changed.

Description

Example

---

dbsave

Save database as CSV file

Syntax

```
List = dbsave(D,FName)
List = dbsave(D,FName,Dates,...)
```

Output arguments

- `List` [cellstr] - - List of actually saved database entries.

Input arguments

- D [struct] - Database whose tseries and numeric entries will be saved.
- FName [char] - Filename under which the CSV will be saved, including its extension.
- Dates [numeric | Inf] Dates or date range on which the tseries objects will be saved.

Options

- 'class=' [true | false] - Include a row with class and size specifications.
- 'comment=' [true | false] - Include a row with comments for tseries objects.
- 'decimal=' [numeric | empty] - Number of decimals up to which the data will be saved; if empty the 'format' option is used.
- 'format=' [char | '%.8e'] - Numeric format that will be used to represent the data, see sprintf for details on formatting, The format must start with a '%', and must not include identifiers specifying order of processing, i.e. the '$' signs, or left-justify flags, the '-' signs.
- 'freqLetters=' [char | 'YHQBM'] - Five letters to represent the five possible date frequencies (annual, semi-annual, quarterly, bimonthly, monthly).
- 'matchFreq=' [true | false] - Save only the tseries whose date frequencies match the input vector of dates, Dates.
- 'nan=' [char | 'NaN'] - String that will be used to represent NaNs.
- 'saveSubdb=' [true | false] - Save sub-databases (structs found within the struct D); the sub-databases will be saved to separate CSF files.
- 'userData=' [char | 'userdata'] - Field name from which any kind of userdata will be read and saved in the CSV file.

Description

The data saved include also imaginary parts of complex numbers.

Saving user data with the database

If your database contains field named 'userdata=', this will be saved in the CSV file on a separate row. The 'userdata=' field can be any combination of numeric, char, and cell arrays and 1-by-1 structs.

You can use the 'userdata=' field to describe the database or preserve any sort of metadata. To change the name of the field that is treated as user data, use the 'userData=' option.

Example

Create a simple database with two time series.

```
d = struct();
d.x = tseries(qq(2010,1):qq(2010,4),@rand);
d.y = tseries(qq(2010,1):qq(2010,4),@rand);
```

Add your own description of the database, e.g.

```
d.userdata = {'My database',datestr(now())};
```

Save the database as CSV using dbsave,

```
dbsave(d,'mydatabase.csv');
```

When you later load the database,

```
d = dbload('mydatabase.csv')

d =

   userdata: {'My database'  '23-Sep-2011 14:10:17'}
          x: [4x1 tseries]
          y: [4x1 tseries]
```

the database will preserve the `'userdata='` field.

Example

To change the field name under which you store your own user data, use the `'userdata='` option when running dbsave,

```
d = struct();
d.x = tseries(qq(2010,1):qq(2010,4),@rand);
d.y = tseries(qq(2010,1):qq(2010,4),@rand);
d.MYUSERDATA = {'My database',datestr(now())};
dbsave(d,'mydatabase.csv',Inf,'userData=','MYUSERDATA');
```

The name of the user data field is also kept in the CSV file so that dbload works fine in this case, too, and returns a database identical to the saved one,

```
d = dbload('mydatabase.csv')

d =

   MYUSERDATA: {'My database'  '23-Sep-2011 14:10:17'}
            x: [4x1 tseries]
            y: [4x1 tseries]
```

---

dbsearchuserdata

Search database to find tseries by matching the content of their userdata fields

Syntax

```
[List,SubD] = dbsearchuserdata(D,Field1,Regexp1,Field2,Regexp2,...)
[List,SubD] = dbsearchuserdata(D,Flag,Field1,Regexp1,Field2,Regexp2,...)
```

Input arguments

- `D` [struct] - Input database whose tseries fields will be searched.
- `Flag` [`'-all'` | `'-any'`] - Specifies if all conditions or any condition must be met for the series to pass the test; if not specified, `'-all'` is assumed.
- `Field1`, `Field2`, ... [char] - Names of fields in the userdata struct.
- `Regexp1`, `Regexp2`, ... [char] - Regular expressions against which the respective userdata fields will be matched.

Output arguments

- `List` [cellstr] - Names of tseries that pass the test.
- `Subd` [struct] - Sub-database with only those tseries that pass the test.

Description

For a successful match, the userdata must be a struct, and the tested fields must be text strings.

Use an equal sign, `=`, after the name of the userdata fields in `Field1`, `Field2`, etc. to request a case-insensitive match, and an equal-shart sign, `=#`, to indiciate a case-sensitive match.

Example

```
[list,dd] = dbsearchuserdata(d,'.DESC=','Exchange rate','.SOURCE=#','IMF');
```

Each individual tseries object in the database D will be tested for two conditions:

- whether its user data is a struct including a field named DESC, and the field contains a string 'Exchange rate' in it (case insensitive, e.g. 'eXcHaNgE rAtE' will also be matched);
- whether its user data is a struct including a field named SOURCE, and the field contains a string 'IMF' in it (case sensitive, e.g. 'Imf' will not be matched).

All tseries object that pass both of these conditions are returned in the List and the output database D.

---

dbsplit

Split database into mutliple databases

Syntax

```
[D1,D2,...,DN,D] = dbsplit(D,Rule1,Rule2,...,RuleN,...)
```

Input arguments

- D [struct] - Input database that will be split.
- Rule1, Rule2, ..., RuleN [cellstr] - Each rule is a 1-by-2 cell array, {testRex,newName}, where testRex is a test regexp pattern to select entries from the input database, D, for inclusion in the K-th output database, and newName is a new name pattern that will be used to name the entry in the output database.

Output arguments

- D1, D2, ..., DN [struct] - Output databases.
- D [struct] - Input database with remaining fields (if 'discard=' true) or the original input database (if 'discard=' false).

Options

- 'discard=' [true | false] - Discard input database entries when they are included in an output database, and do not re-use them in other output databases; if false, an input database entry can occur in more than one output databases.

Description

The test regexp pattern and the new name pattern in each rule work as an expression-replace pair in regexprep – see doc regexprep. The test patterns is a regexp string where you can capture tokens (...) for use in the new name pattern, $1, $2, etc.

Example

The database D contains time series for two regions, US and EU:

```
D =
    US_GDP: [40x1 tseries]
    US_CPI: [40x1 tseries]
    EU_GDP: [40x1 tseries]
    EU_CPI: [40x1 tseries]
```

We split the database into two separate databases, one with US data only, the other with EU data only. We also strip the time series names of the country prefixes in the new databases.

```
[US,EU,DD[] = dbsplit(D,{'^US_(.*)','$1'},{'^EU_(.*)','$1'})

US =
    GDP: [40x1 tseries]
    CPI: [40x1 tseries]
EU =
    CPI: [40x1 tseries]
    GDP: [40x1 tseries]
DD =
struct with no fields.
```

---

dbuserdatalov

List of values found in a specified user data field in tseries objects

Syntax

```
LOV = dbuserdatalov(D,FIELD)
```

Input arguments

- D [struct] - Input database whose tseries objects will be searched.
- FIELD [char] - Name of a userdata field whose values will be collected across all tseries objects.

Output arguments

- LOV [cellstr] - List of values found in the field FIELD of all tseries objects; only char values (text strings) are included; each value is included only once in LOV.

Description

Example

---

minus

Remove entries from a database

Syntax

```
D = D - List
```

Input arguments

- D [struct] - Input database from which some entries will be removed.
- List [char | cellstr] - List of entries that will be removed from D.

Output arguments

- D [struct] - Output database with entries listed in List removed from it.

Description

This functino works the same way as the built-in function `rmfield` except it does not throw an error when some of the entries listed in `List` are not found in `D`.

Example

---

mtimes

Keep only the database entries that are on the list

Syntax

```
D = D * List
```

Input arguments

- `D` [struct] - Input database.
- `List` [cellstr] - List of entries that will be kept in the output database.

Output arguments

- `D` [struct] - Output database where only the input entries that are in the `List` are included.

Description

Example

---

plus

Merge entries from two databases together

Syntax

```
D = D1 + D2
```

Input arguments

- D1 [struct] - First input database.
- D2 [struct] - Second input database.

Output arguments

- D [struct] - Output database with entries from both input database; if the same entry name exists in both databases, the second database is used.

Description

Example

---

xls2csv

Convert XLS file to CSV file

Syntax

```
xls2csv(InpFile)
xls2csv(InpFile,OutpFile,...)
```

Input arguments

- InpFile [char] - Name of an XLS input file that will be converted to CSV.
- OutpFile [empty | char] - Name of the CSV output file; if not supplied or empty, the CSV file name will be derived from the XLS input file name.

Options

- 'sheet=' [numeric | char | 1] - Worksheet in the XLS file that will be saved; can be either the sheet number or the sheet name.

Description

This function calls a third-party JavaScript (courtesy of Christopher West). The script uses an MS Excel application on the background, and hence MS Excel must be installed on the computer.

Only one worksheet at a time can be saved to CSV. By default, it is the first worksheet found in the input XLS file; use the option 'sheet=' to control which worksheet will be saved.

See also $irisroot/+thirdparty/xls2csv.js for copyright information.

Example

Save the first worksheets of the following XLS files to CSV files.

```
xls2csv('myDataFile.xls');
xls2csv('C:\Data\myDataFile.xls');
```

Example

Save the worksheet named 'Sheet3' to a CSV file; the name of the CSV file will be 'myDataFile.csv'.

```
xls2csv('myDataFile.xls',[],'sheet=','Sheet3');
```

Example

Save the second worksheet to a CSV file under the name 'myDataFile_2.csv'.

```
xls2csv('myDataFile.xls','myDataFile_2.csv,'sheet=',2);
```

Part V —
Reporting and Publishing

21 PDF Reports (report Package and Objects)

New report

- new P450 - Create new empty report object.
- copy P433 - Create a copy of a report object.

Compiling PDF report

- publish P451 - Compile PDF from report object.

Top-level objects

- table P459 - Start new table.
- figure P437 - Start new figure.
- userfigure P464 - Insert existing figure window.
- matrix P444 - Insert matrix or numeric array.
- modelfile P448 - Write formatted model file.
- array P429 - Insert array with user data.
- tex P463 - Include LaTeX code or verbatim input in report.

Inspecting and maninpulating report objects

- disp P434 - Display the structure of report object.
- display P434 - Display the structure of report object.
- findall P439 - Find all objects of a given type within report object.

Figure objects

- graph P440 - Add graph to figure.

Table and graph objects

- band P432 - Add new data with lower and upper bounds to graph or table.
- fanchart P436 - Add fanchart to graph.
- series P454 - Add new data to graph or table.
- subheading P458 - Enter subheading in table.
- vline P465 - Add vertical line to graph.
- highlight P442 - Highlight range in graph.

Structuring reports

- align P427 - Vertically align the following K objects.
- empty P435 - Empty report object.
- include P443 - Include text or LaTeX input file in the report.
- merge P448 - Merge the content of two or more report objects.
- pagebreak P450 - Force page break.
- section P453 - Start new section in report.

Getting on-line help on report functions

```
help report
help report/function_name
```

Generic options

The following generic options can be used on any of the report objects.

- 'inputFormat=' [*'plain' | 'latex'] - Input format for user supplied text strings (such as captions, headings, footnotes, etc); 'latex' means they are assumed to be valid LaTeXstrings, and will be inserted straight into the report code with no modification.
- 'saveAs=' [char | empty] - (Not inheritable from parent objects) Save the LaTeX code generated for the respective report element in a text file under the specified name.

■ align

Vertically align the following K objects

Syntax

```
P.align(Caption,K,NCol,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.

- Caption [char] - Caption displayed only when describing the structure of the report on the screen, but not in the actual PDF report.
- K [numeric] - Number of objects following this align that will be vertically aligned.
- NCol [numeric] - Number of columns in which the objects will vertically aligned.

Options

- 'hspace=' [numeric | 2] - Horizontal space (in em units) inserted between two neighbouring objects.
- 'separator=' [char | '\medskip\par'] - (Inheritable from parent objects) LaTeX commands that will be inserted after the aligned objects.
- 'shareCaption=' ['auto' | true | false] - (Inheritable from parent objects) Place a shared caption (title and subtitle) over each row of objects; the title of the first object in each row is used; 'auto' means that the caption will be shared if they are identical for all objects in a row.
- 'typeface=' [char | empty] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the align element as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

Description

Vertically aligned can be the following types of objects:

- figure P437
- table P459
- matrix P444
- array P429

Note that the align object itself has no caption (even if you specify one it will not be used). Only the objects within align will be given captions. If the objects aligned on one row have identical captions (i.e. both titles and subtitles), only one caption will be displayed centred above the objects.

Because empty P435 objects count in the total number of objects inluded in align, you can use empty P435 in to create blank space in a particular position.

Example

array

Insert array with user data

Syntax

```
P.array(Caption,Data)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char | cellstr] - Title or a cell array with title and subtitle displayed at the top of the array; see Description for splitting the title or subtitle into multiple lines.
- Data [cell] - Cell array with input data; numeric and text entries are allowed.

Options

- 'arrayStretch=' [numeric | 1.15] - (Inherited) Stretch between lines in the array (in pts).
- 'captionTypeface=' [cellstr | char | '\large\bfseries'] - (Inherited) LaTeX format commands for typesetting the array caption; the subcaption format can be entered as the second cell in a cell array.
- 'colWidth=' [numeric | NaN] - (Inheritable from parent objects) Width, or a vector of widhts, of the array columns in emunits; NaN means the width of the column will adjust automatically.
- 'format=' [char | '%.2f'] - (Inherited) Numeric format string; see help on the built-in sprintf function.
- 'footnote=' [char | empty] - (Inherited) Footnote at the array title; only shows if the title is non-empty.
- 'heading=' [char | cellstr | empty] - (Inherited) User-supplied heading, i.e. an extra row or rows at the top of the array. The heading can be either a LaTeX code, or a cell array whose size is consistent with Data. The heading is repeated at the top of each new page when used with 'long=' true.
- 'inf=' [char | '∞'] - (Inherited) LaTeX string that will be used to typeset Infs.
- 'long=' [true | false] - (Inherited) If true, the array may stretch over more than one page.
- 'longFoot=' [char | empty] - (Inherited) Footnote that appears at the bottom of the array (if it is longer than one page) on each page except the last one; works only with 'long=' true.

- 'longFootPosition=' ['centre' | 'left' | 'right'] - (Inherited) Horizontal alignment of the footnote in long arrays; works only with 'long=' true.
- 'nan=' [char | '$\cdots$'] - (Inherited) LaTeX string that will be used to typeset NaNs.
- 'pureZero=' [char | empty] - (Inherited) LaTeX string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- 'printedZero=' [char | empty] - (Inherited) LaTeX string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.
- 'separator=' [char | '\medskip\par'] - (Inherited) LaTeX commands that will be inserted after the array.
- 'sideways=' [true | false] - (Inherited) Print the array rotated by 90 degrees.
- 'tabcolsep=' [NaN | numeric] - (Inherited) Space between columns in the array, measured in em units; NaN means the LaTeX default.
- 'typeface=' [char | empty] - (Not inherited) LaTeX code specifying the typeface for the array as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

Generic options

See help on generic options P426 in report objects.

Description

The input cell array Data can contain either strings or numeric values, or horizontal rules. Numeric values are printed using the standard sprintf function and formatted using the 'format=' option. Horizontal rules must be entered as a string of five (or more) dashes, '-----', in the first cell of the respective row, with all other cells empty in that row. If you wish to include a LaTeX command or a piece of LaTeX code, you must enclose it in curly brackets.

Titles and subtitles

The input argument Caption can be either a text string, or a 1-by-2 cell array of strings. In the latter case, the first cell will be printed as a title, and the second cell will be printed as a subtitle.

To split the title or subtitle into multiple lines, use the following LaTeX commands wrapped in curly brackets: {\\} or {\\[Xpt]}, where X is the width of an extra vertical space (in points) added between the respective lines.

Example

These commands create a table with two rows separated by a horizontal rule, and three columns in each of them. The middle columns will have Greek letters printed in LaTeX math mode.

```
x = report.new();

A = { ...
    'First row','{\$\textbackslash alpha\$}',10000; ...
    '-----','',''; ...
    'Second row','{\$\textbackslash beta\$}',20000; ...
};

x.array('My Table',A);

x.publish('test1.pdf');

open test1.pdf;
```

Example

Use the option `'inputFormat='` to change the way the input strings are interpreted. Compare the two tables in the resulting PDF.

```
x = report.new();

A = { ...
    1,2,3; ...
    '$\alpha$','b','c', ...
    };

x.array('Table with Plain Input Format (Default)',A, ...
    'heading=',{'A','B','$\Gamma$';'-----','',''});

x.array('Table with LaTeX Input Format',A, ...
    'heading=',{'A','B','$\Gamma$';'-----','',''}, ...
    'inputFormat=','latex');

x.publish('test2.pdf');

open test2.pdf;
```

band

Add new data with lower and upper bounds to graph or table

Syntax

```
P.band(Caption,X,Low,High,...)
```

Input arguments

- `P` [struct] - Report object created by the `report.new` P450 function.
- `Caption` [char] - Caption used as a default legend entry in a graph, or in the leading column in a table.
- `X` [tseries] - Input data with the centre of the band.
- `Low` [tseries] - Input data with lower bounds; can be specified either relative to the centre or absolute, see the option `'relative='`.
- `High` [tseries] - Input data with upper bounds; can be specified either relative to the centre or absolute, see the option `'relative='`.

Generic options

See help on generic options P426 in report objects.

Options for table and graph bands

- `'excludeFromLegend='` [`true` | `false`] - Exclude bands around central lines from legend.
- `'high='` [char | 'High'] - (Inheritable from parent objects) Mark used to denote the upper bound.
- `'low='` [char | 'Low'] - (Inheritable from parent objects) Mark used to denote the lower bound.
- `'relative='` [`true` | `false`] - (Inheritable from parent objects) If true, the data for the lower and upper bounds are relative to the centre, i.e. the bounds will be added to the centre (in this case, `LOW` must be negative numbers and `HIGH` must be positive numbers). If false, the bounds are absolute data (in this case `LOW` must be lower than `X`, and `HIGH` must be higher than `X`).

Options for table bands

- 'bandTypeface=' [char | '\footnotesize'] - (Inheritable from parent objects) LaTeX format string used to typeset the lower and upper bounds.%

Options for graph bands

- 'plotType=' ['errorbar' | 'patch'] - Type of plot used to draw the band.
- 'relative=' [true | false] - (Inheritable from parent objects) If true the lower and upper bounds will be, respectively, subtracted from and added to to the middle line.
- 'white=' [numeric | 0.85] - (Inheritable from parent objects) Proportion of white colour mixed with the center line color and used to fill the band area.

See help on report/series P454 for other options available.

Description

Example

---

copy

Create a copy of a report object

Syntax

```
Q = copy(P)
```

Input arguments

- P [report] - Report object whose copy will be created.

Output arguments

- Q [report] - Copy of the input report object.

Description

Because report is a handle class object, a plain assignment

```
Q = P;
```

creates a handle to the same copy of a report object. In other words, changes in Q will also change P and vice versa. To make a new, independent copy of an existing report object, you need to run

```
Q = copy(P);
```

---

disp

Display the structure of report object

Syntax

```
disp(X)
```

Input arguments

- X [report] - Report object.

Description

Example

---

disp

Display the structure of report object

Syntax

```
X
```

Input arguments

- X [report] - Report object.

Description

Example

---

empty

Empty report object

Syntax

```
P.empty()
P.empty(Caption,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char] - Caption for the empty objet; the caption is only displayed in the on-screen report structure.

Generic options

See help on generic options P426 in report objects.

Description

The empty object does not produce any visible output in the report. It can be used in align P427 or figure P437 to create blank space.

Example

---

fanchart

Add fanchart to graph

Syntax

```
P.fanchart(Cap,X,Std,Prob,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Cap [char] - Caption used as a legend entry for the line (mean of fanchart)
- X [tseries] - Tseries object with input data to be displayed.
- Std [tseries] - Tseries object with standard deviations of input data.
- Prob [numeric] - Confidence porbabilities of intervals to be displayed.

Options for fancharts

- 'asym=' [numeric | tseries | 1] - Ratio of asymmetry (area of upper part to one of lower part).
- 'exclude=' [numeric | true | false] - Exclude some of the confidence intervals.
- 'factor=' [numeric | 1] - factor to increase or decrease input standard deviations
- 'fanLegend=' [cell | NaN | Inf] - Legend entries used instead of confidence interval values; Inf means all confidence intervals values will be used to construct legend entries; NaN means the intervals will be exluded from legend; NaN in cellstr means the intervals of respective fancharts will be exluded from legend.

See help on report/series P454 for other options available.

Description

The confidence intervals are based on normal distributions with standard deviations supplied by the user. Optionally, the user can also specify assumptions about asymmetry and/or common correction factors.

Example

---

figure

Start new figure

Syntax

```
P.figure(Caption,...)
```

Syntax to capture an existing figure window

This is an obsolete syntax, and will be removed from IRIS in a future release. Use report/userfigure P464 instead.

```
P.figure(Caption,H,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char | cellstr] - Title or a cell array with title and subtitle displayed at the top of the figure; see Description for splitting the title or subtitle into multiple lines.
- H [numeric] - See help on report/userfigure P464.

Options

- 'aspectRatio=' [@auto | numeric] - Plot box aspect ratio for all graphs in the figure; must be a 1-by-2 vector describing the horizontal-to-vertical ratio.
- 'captionTypeface=' [cellstr | char | '\large\bfseries'] - LaTeX format commands for typesetting the figure caption; the subcaption format can be entered as the second cell in a cell array.
- 'close=' [true | false] - (Inheritable from parent objects) Close the underlying figure window when finished; see Description.

- 'separator=' [char | '\medskip\par'] - (Inheritable from parent objects) LaTeX commands that will be inserted after the figure.
- 'figureOpt=' [cell | empty] - Figure options that will be applied to the figure handle at opening.
- 'figureScale=' [numeric | 0.85] - (Inheritable from parent objects) Scale of the figure in the LaTeX document.
- 'figureTrim=' [numeric | 0] - Trim figure when it is inserted into the report by the specified amount of points; must be either a scalar or a 1-by-4 vector (points removed from left, bottom, right, top).
- 'footnote=' [char | empty] - Footnote at the figure title; only shows if the title is non-empty.
- 'sideways=' [true | false] - (Inheritable from parent objects) Print the table rotated by 90 degrees.
- 'style=' [struct | empty] - Apply this cascading style structure to the figure; see grfun.style P484.
- 'subplot=' [numeric | 'auto'] - (Inheritable from parent objects) Subplot division of the figure.
- 'typeface=' [char | empty] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the figure as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).
- 'visible=' [true | false] - (Inheritable from parent objects) Visibility of the underlying Matlab figure window.

Generic options

See help on generic options P426 in report objects.

Description

Figures are top-level report objects and cannot be nested within other report objects, except align P427. Figure objects can have the following types of children:

- graph P440;
- empty P435.

Titles and subtitles

The input argument `Caption` can be either a text string, or a 1-by-2 cell array of strings. In the latter case, the first cell will be printed as a title, and the second cell will be printed as a subtitle.

To split the title or subtitle into multiple lines, use the following LaTeX commands wrapped in curly brackets: `{\\}` or `{\\[Xpt]}`, where `X` is the width of an extra vertical space (in points) added between the respective lines.

Figure handle

If the option `'close='` is set to `false` the figure window will remain open after the report is published. The handle to this figure window will be included in the field `.figureHandle` of the information struct `Info` returned by `report/publish` P451.

Example

■ findall

Find all objects of a given type within report object

Syntax

```
Obj = findall(X,Type1,Type2,...)
```

Input arguments

- `X` [report] - Report object.
- `Type1`, `Type2` [char] - Names of report objects that will be looked for in report `X`.

Output arguments

- `Obj` [cell] - Cell array of all objects of the give type(s) found in report `X`.

Description

Example

---

graph

Add graph to figure

Syntax

```
P.graph(Cap,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Cap [char | cellstr | @auto] - Title, or cell array with title and subtitle, displayed at the top of the graph; @auto means that the first comment from the first child series object will be used for the title.

Options

- 'axesOptions=' [cell | empty] - (Inheritable) Options executed by calling set on the axes handle before running 'postProcess='.
- 'dateTick=' [numeric | Inf] - (Inheritable) Date tick spacing.
- 'grid=' [@auto | true | false] - (Inheritable) Display grid lines; if @auto, 'grid=' is true unless a right-hand-side axis is plotted.
- 'legend=' [false | true] - (Inheritable) Add legend to the graph.
- 'legendLocation=' [char | 'best' | 'bottom'] - (Inheritable) Location of the legend box; see help on legend for values available.
- 'postProcess=' [char | empty] - (Inheritable) String with Matlab commands executed after the graph has been drawn and styled; see Description.
- 'preProcess=' [char | empty] - (Inheritable) String with Matlab commands executed before the graph has been drawn and styled; see Description.
- 'range=' [numeric | Inf] - (Inheritable) Graph range.
- 'rhsAxesOptions=' [cell | empty] - (Inheritable) Options executed by calling set on the RHS axes handle before running 'postProcess='.
- 'style=' [struct | empty] - (Inheritable) Apply this style structure to the graph and its children; see help on grfun.style P484.

- 'tight=' [@auto | true | false] - (Inheritable) Set the y-axis limits to the minimum and maximum of displayed data; if @auto, 'tight=' is true unless a right-hand-side axis is plotted.
- 'xLabel=' [char | empty] - Label the x-axis.
- 'yLabel=' [char | empty] - Label the y-axis.
- 'zeroLine=' [true | false | cell] - (Inheritable) Add a horizontal zero line if zero is included on the y-axis; specify zeroline options in a cell array.

Date format options

See dat2str P280 for details on date format options.

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months.
- 'standinMonth=' [numeric | 'last' | 1] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.

Generic options

See help on generic options P426 in report objects.

Description

The options 'preProcess=' and 'postProcess=' give you additional flexibility in customising the graphics style of the axes object. The values assigned to these options are expected to be strings with an executable Matlab command, or commands separated with semi-colons (as if typed on one line in the command window). The command can refer to the following variables:

- H - a handle to the currently processed axes object.
- L - a handle to the corresponding legend object; if no legend object exists for the axes H, L will be NaN.

Example

Create a one-page report with a chart in on the LHS and the legend moved to the RHS. Use the function grfun.movetosubplot in the option 'postProcess=', referring to L (handle to the legend object associated with the respective axes object) to move the legend around.

```
% Create random data series.
A = tseries(1:10,@rand);
B = tseries(1:10,@rand);

% Open a new report.
x = report.new();

% Open a new figure in the report with a 1-by-2 layout.
x.figure('My Figure','subplot=',[1,2]);

    % The graph will be placed in the LHS space.
    % Use `grfun.movetosubplot` to move the legend to the RHS space.
    x.graph('My Graph','legend=',true, ...
        'postProcess=','grfun.movetosubplot(L,1,2,2)');

        x.series('Series A',A);
        x.series('Series B',B);

x.publish('test.pdf');
open test.pdf;
```

---

highlight

Highlight range in graph

Syntax

```
P.highlight(Caption,Range,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.

- Caption [char] - Caption used to annotate the highlighted area.
- Range [cell | numeric] - Date range, or a cell array of ranges, that will be highlighted.

Options

- 'hPosition=' ['centre' | 'left' | 'right'] - (Inheritable from parent objects) Horizontal position of the caption.
- 'vPosition=' ['bottom' | 'middle' | 'top'] - (Inheritable from parent objects) Vertical position of the caption relative to the edges of the highlighted area.

Generic options

See help on generic options P426 in report objects.

Description

Example

---

include

Include text or LaTeX input file in the report

Syntax

```
P.include(Caption,FileName,...)
```

Input arguments

- P [struct] - Report object created by the function report.new P450.
- Caption [char] - Caption displayed at the top of the file included.
- FileName [char] - File name that will be included here.

Options

- 'centering=' [true | false] - (Inheritable from parent objects) Centre the content of the file on the page.
- 'separator=' [char | empty] - (Not inheritable from parent objects) LaTeX commands that will be inserted after the table.
- 'typeface=' [char | empty] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the include element as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).
- 'verbatim=' [true | false] - (Not inheritable from parent objects) Enclose the content of the file in a verbatim environment.

Generic options

See help on generic options P426 in report objects.

Description

Example

---

matrix

Insert matrix or numeric array

Syntax

```
P.matrix(Caption,Data,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char | cellstr] - Title or a cell array with title and subtitle displayed at the top of the matrix; see Description for splitting the title or subtitle into multiple lines.
- Data [numeric] - Numeric array with input data.

Options

- 'arrayStretch=' [numeric | 1.15] - (Inheritable from parent objects) Stretch between lines in the matrix (in pts).
- 'captionTypeface=' [cellstr | char | *''*] - LaTeX format commands for typesetting the matrix caption; the subcaption format can be entered as the second cell in a cell array.
- **'colNames=' [cellstr | empty] - (Inheritable from parent objects) Names for individual matrix columns, displayed at the top of the matrix.**
- **'colWidth=' [numeric | NaN] - (Inheritable from parent objects) Width, or a vector of widhts, of the matrix columns in emunits; NaN means the width of the column will adjust automatically.**
- **'condFormat=' [struct | empty] - (Inheritable from parent objects) Structure with .test and .format fields describing conditional formatting of individual matrix entries.**
- **'footnote=' [char | empty] - Footnote at the matrix title; only shows if the title is non-empty.**
- **'format=' [char | '%.2f'] - (Inheritable from parent objects) Numeric format string; see help on the built-in sprintf function.**
- **'heading=' [char | empty] - (Inheritable from parent objects) User-supplied heading, i.e. an extra row or rows at the top of the matrix.**
- **'inf=' [char | '∞'] - (Inheritable from parent objects) LaTeX string that will be used to typeset Infs.**
- **'long=' [true | false] - (Inheritable from parent objects) If true, the matrix may stretch over more than one page.**
- **'longFoot=' [char | empty] - (Inheritable from parent objects) Works only with 'long=' true: Footnote that appears at the bottom of the matrix (if it is longer than one page) on each page except the last one.**

- **'longFootPosition='** [**'centre'** | **'left'** | **'right'**] - (Inheritable from parent objects) Works only with **'long='** **true**: Horizontal alignment of the footnote in long matrices.
- **'nan='** [char | **'$\cdots$'**] - (Inheritable from parent objects) LaTeX string that will be used to typeset **NaN**s.
- **'pureZero='** [char | empty] - (Inheritable from parent objects) LaTeX string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- **'printedZero='** [char | empty] - (Inheritable from parent objects) LaTeX string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.
- **'rotateColNames='** [**true** | **false** | numeric] - Rotate the names of columns by the specified number of degrees; **true** means rotate by 90 degrees.
- **'rowNames='** [cellstr | empty] - (Inheritable from parent objects) Names fr individual matrix rows, displayed left of the matrix.
- **'separator='** [char | **'\medskip\par'**] - (Inheritable from parent objects) LaTeX commands that will be inserted after the matrix.
- **'sideways='** [**true** | **false**] - (Inheritable from parent objects) Print the matrix rotated by 90 degrees.
- **'tabcolsep='** [**NaN** | numeric] - (Inheritable from parent objects) Space between columns in the matrix, measured in em units; **NaN** means the LaTeX default.
- **'typeface='** [char | empty] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the matrix as a whole; it must use the declarative forms (such as **\itshape**) and not the command forms (such as **\textit{...}**).

Generic options

See help on generic options P426 in report objects.

Description

Conditional formatting

The conditional format struct (or an array of structs) specified through the 'condFormat=' option must have two fields, .test and .format.

The .test field is a text string with a Matlab expression. The expression must evaluate to a scalar true or false, and can refer to the following attributes associated with each entry in the data part of the matrix:

- value - the numerical value of the entry;
- row - the row number within the data part of the matrix;
- col - the column number within the data part of the matrix;
- rowname - the row name right of which the entry appears;
- colname - the column name under which the entry appears;
- rowvalues - a row vector of all values in the current row;
- colvalues - a column vector of all values in the current column;
- allvalues - a matrix of all values.

You can combine a number of attribues within one test, using the logical operators, e.g.

```
value > 0 && row > 3
value == max(rowvalues) && strcmp(rowname,'x')
```

The .format fields of the conditional format structure consist of LaTeX commands that will be use to typeset the corresponding entry. The reference to the entry itself is through a question mark. The entries are typeset in math mode; this for instance meanse that for bold or italic typface, you must use the \mathbf{...} and \mathit{...} commands.

In addition to standard LaTeX commands, you can use the following IRIS commands in the format strings:

- \sprintf{FFFF} - to modify the way each numeric entry that passes the test is printed by the sprintf function; FFFF is one of the standard sprintf formattting strings.

You can combine multiple tests and their correponding formats in one structure; they will be all applied to each entry in the specified order.

Titles and subtitles

The input argument Caption can be either a text string, or a 1-by-2 cell array of strings. In the latter case, the first cell will be printed as a title, and the second cell will be printed as a subtitle.

To split the title or subtitle into multiple lines, use the following LaTeX commands wrapped in curly brackets: {\\} or {\\[Xpt]}, where X is the width of an extra vertical space (in points) added between the respective lines.

Example

---

merge

Merge the content of two or more report objects

Syntax

```
P.merge(P1,P2,...)
```

Input arguments

- P [report] - Report object created by the report.new P450 function.
- P1, P2 [report] - Other report objects whose contents will be added to P in order of appearance.

Description

Example

---

modelfile

Write formatted model file

Syntax

```
P.modelfile(Caption,FileName,...)
P.modelfile(Caption,FileName,M,...)
```

Input arguments

- `P` [report] - Report object created by the `report.new` P450 function.
- `Caption` [char | cellstr] - Title and subtitle displayed at the top of the table.
- `FileName` [char] - Model file name.
- `M` [model] - Model object from which the values of parameters and std devs of shocks will be read; if missing no parameter values or std devs will be printed.

Options

- `'latexAlias='` [`true` | `false`] - Treat alias in labels as LaTeX code and typeset it that way.
- `'lines='` [numeric | `@all`] - Print only selected lines of the model file `FileName`; `@all` means all lines will be printed.
- `'lineNumbers='` [`true` | `false`] - Display line numbers.
- `'footnote='` [char | empty] - Footnote at the model file title; only shows if the title is non-empty.
- `'paramValues='` [`true` | `false`] - Display the values of parameters and std devs of shocks next to each occurence of a parameter or a shock; this option works only if a model object `M` is entered as the 3rd input argument.
- `'syntax='` [`true` | `false`] - Highlight model file syntax; this includes model language keywords, descriptions of variables, shocks and parameters, and equation labels.
- `'typeface='` [char | empty] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the model file as a whole; it must use the declarative forms (such as `\itshape`) and not the command forms (such as `\textit{...}`).

Description

If you enter a model object with multiple parameterisations, only the first parameterisation will get reported.

At the moment, the syntax highlighting in model file reports does not handle correctly comment blocks, i.e. `%{ ... %}`.

Example

new

Create new empty report object

Syntax

```
P = report.new(Cap,...)
```

Output arguments

- P [struct] - Report object with function handles through wich the individual report elements can be created.
- Cap [char] - Report caption; the caption will also be printed on the title page of the report if published with the option 'makeTitle=' true.

Options

- 'centering=' [true | false] - All report elements, except tex P463, will be centered on the page.
- 'orientation=' ['landscape' | 'portrait'] - Paper orientation of the published report.

Report options are cascading. You can specify any of an object's options in any of his parent (or ascendant) objects.

---

pagebreak

Force page break

Syntax

```
P.pagebreak(Caption,...)
```

Input arguments

- P [report] - Report object created by the report.new P450 function.
- Caption [char] - Caption for the pagebreak objet; the caption only displays in the on-screen report structure.

Generic options

See help on generic options P426 in report objects.

Description

Example

---

publish

Compile PDF from report object

Syntax

```
[OutpFile,Info] = P.publish(InpFile,...)
```

Input arguments

- P [struct] - Report object created by the report.new function.
- InpFile [char] - File name under which the compiled PDF will be saved.

Output arguments

- OutpFile [char] - Name of the resulting PDF.
- Info [struct] - Information struct with details of building the PDF report; see Description.

Options

- 'abstract=' [char | empty] - Abstract that will displayed on the title page.
- 'abstractWidth=' [numeric | 1] - Width of the abstract on the page as a percentage of the full default width (between 0 and 1).
- 'author=' [char | empty] - List of authors on the title page separated with \and or \\.
- 'cleanup=' [true | false] - Delete all temporary files created when compiling the report.

- 'compile=' [true | false] - Compile the source files to an actual PDF; if false only the source files are created.
- 'date=' [char | '\today'] - Date on the title page.
- 'display=' [true | false] - Display the LaTeXcompiler report on the final iteration.
- 'echo=' [true | false] - If true, the optional flag '-echo' will be used in the Matlab function system when compiling the PDF; this causes the screen output and all prompts to be displayed for each run of the compiler.
- 'epsToPdf=' [char | Inf] - Command line arguments for EPSTOPDF; Inf means OS-specific arguments are used.
- 'fontEnc=' [char | 'T1'] - LaTeX font encoding.
- 'makeTitle=' [true | false] - Produce title page (with title, author, date, and abstract).
- 'package=' [char | cellstr | empty] - Package or list of packages that will be imported in the preamble of the LaTeX file.
- 'paperSize=' ['a4paper' | 'letterpaper'] - Paper size.
- 'orientation=' ['landscape' | 'portrait'] - Paper orientation.
- 'preamble=' [char | empty] - LaTeX commands that will be placed in the LaTeX file preamble.
- 'timeStamp=' [char | 'datestr(now())'] - String printed in the top-left corner of each page.
- 'tempDir=' [char | function_handle | tempname(pwd())] - Directory for storing temporary files; the directory is deleted at the end of the execution if it's empty.
- 'maxRerun=' [numeric | 5] - Maximum number of times the LaTeX compiler will be run to resolve cross-references, etc.
- 'minRerun=' [numeric | 1] - Minimum number of times the LaTeX compiler will be run to resolve cross-references, etc.
- 'textScale=' [numeric | 0.8] - Percentage of the total page area that will be used; the value can be either a scalar (the same percentage for the width and the height) or a 1-by-2 vector (the width and the height).

Description

Difference between 'display=' *and* 'echo='

There are two differences between these otherwise similar options:

- When publishing the final PDF, the PDFLaTeX compiler may be called more than once to resolve cross-references, the table of contents, and so on. Setting 'display=' true only displays the screen output from the final iteration only, while 'echo=' true displays the screen outputs from all iterations.
- In the case of a compiler error unrelated to the LaTeX code, the compiler may stop and prompt the user to respond. The prompt only appears on the screen when 'echo=' true. Otherwise, Matlab may remain in a busy state with no on-screen information, and Ctrl+C may be needed to regain control.

Information struct

The second output argument, Info, is a struct with details of building the PDF report. It contains the following fields:

- .latexRun – the total number of LaTeX compiler runs needed to resolve cross-references and other dependencies;
- .figureHandle – a vector of figure window handles created during the report production process, and not closed (i.e. still existing in the Matlab workspace); to keep figure windows open, use the figure object option 'close=' false. If all figure and userfigure objects inside a report have 'close=' true then Info.figureHandle will be empty.
- .tempDir – empty unless publish is called with 'cleanup=' false; in that case, this is the name of a temporary directory in which all files are saved necessary to build the output PDF are saved.
- .tempFile – empty unless publish is called with 'cleanup=' false; in that case, this is the list of all files (saved in the temporary directory) necessary to build the output PDF.

Example

---

section

Start new section in report

Syntax

```
P.section(CAP,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- CAP [char] - Section title.

Options

- 'numbered=' [true | false] - (Inherited) Numbered section.
- 'separator=' [char | empty] - (Not inherited) LaTeX commands that will be inserted after the table.

Generic options

See help on generic options P426 in report objects.

Description

Example

---

series

Add new data to graph or table

Syntax

```
P.series(Cap,X,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Cap [char | cellstr | @auto] - Caption used as a default legend entry in a graph, or in the leading column in a table; @auto means that the first comment from the input tseries object, X, will be used for the title.
- X [tseries] - Input data that will be added to the current table or graph.

Options for both table series and graph series

- 'marks=' [cellstr | empty] - (Inheritable from parent objects) Marks that will be added to the legend entries in graphs, or printed in a third column in tables, to distinguish the individual columns of possibly multivariate input tseries objects.
- 'showMarks=' [true | false] - (Inheritable from parent objects) Use the marks defined in the 'marks=' option to label the individual rows when input data is a multivariate tseries object.

Options for table series

- 'autoData=' [function_handle | cell | empty] - Function, or a cell array of functions, that will be used to produce new columns in the input tseries object (i.e. new rows of ouput in the report).
- 'condFormat=' [struct | empty] - (Inheritable from parent objects) Structure with .test and .format fields describing conditional formatting of individual table entries.
- 'decimal=' [numeric | NaN] - (Inheritable from parent objects) Number of decimals that will be displayed; if NaN the 'format=' option is used instead.
- 'format=' [char | '%.2f'] - (Inheritable from parent objects) Numeric format string; see help on the built-in sprintf function.
- 'footnote=' [char | empty] - Footnote at the series text.
- 'highlight=' [numeric | empty] - Periods for which the data entries will highlighted.
- 'inf=' [char | '\ensuremath{\infty}'] - (Inheritable from parent objects) LaTeX string that will be used to typeset Inf entries.
- 'nan=' [char | '\ensuremath{\cdot}'] - (Inheritable from parent objects) LaTeX string that will be used to typeset NaN entries.
- 'pureZero=' [char | empty] - (Inheritable from parent objects) LaTeX string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- 'printedZero=' [char | empty] - (Inheritable from parent objects) LaTeX string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.
- 'rowHighlight=' [true | false] - Highlight the entire row, including the text, units and marks at the beginnig; because of a bug in the LaTex package colortbl, this option cannot be combined with the option 'highlight=' in report/table P459.

- 'separator=' [char | empty] - LaTeX commands that will be inserted immediately after the end of the table row, i.e. appended to \, within a tabular mode.
- 'units=' [char] - (Inheritable from parent objects) Description of input data units that will be displayed in the second column of tables.

Options for graph series

- 'legendEntry=' [char | cellstr | NaN | @auto] - Legend entries used instead of the series caption and marks; @auto means the caption and marks will be used to construct legend entries; NaN means the series will be exluded from legend.
- 'plotFunc=' [@area | @bar | @barcon | @plot | @plotcmp | @plotpred | @stem] - (Inheritable from parent objects) Plot function that will be used to create graphs.
- 'plotOptions=' [cell | empty] - Options passed as the last input arguments to the plot function.
- yAxis=' ['left' | *'right'] - Choose the LHS or RHS axis to plot this series; see also comments on LHS-RHS plots in Description.

Generic options

See help on generic options P426 in report objects.

Description

Using the options 'nan=', 'inf=', 'pureZero=' *and* 'printedZero='

When specifying the LaTeX string for these options, bear in mind that the table entries are printed in the math model. This means that whenever you wish to print a normal text, you need to use an appropriate text formatting command allowed within a math mode. Most frequently, it would be '\textnormal{...}'.

Using the option 'plotFunc='

When you set the option to 'plotpred', the input data X (second input argument) must be a multicolumn tseries object where the first column is the time series observations, and the second and further columns are its Kalman filter predictions as returned by the filter function.

Conditional formatting

The conditional format struct (or an array of structs) specified through the 'condFormat=' option must have two fields, .test and .format.

The .test field is a text string with a Matlab expression. The expression must evaluate to a scalar true or false, and can refer to the following attributes associated with each entry in the data part of the table:

- value - the numerical value of the entry,
- date - the date under which the entry appears,
- year - the year under which the entry appears,
- period - the period within the year (e.g. month or quarter) under which the entry appears,
- freq - the frequency of the date under which the entry appears,
- text - the text label on the left,
- mark - the text mark on the left used to describe the individual rows reported for multivariate series,
- row - the row number within a multivariate series.
- rowvalues - a row vector of all values on the current row.

If the table is based on user-defined structure of columns (option 'colstruct=' in table P459), the following additional attributes are available

- colname - descriptor of the column (text in the headline).

You can combine a number of attribues within one test, using the logical operators, e.g.

```
'value > 0 && year > 2010'
```

The .format fields of the conditional format structure consist of LaTeX commands that will be use to typeset the corresponding entry. The reference to the entry itself is through a question mark. The entries are typeset in math mode; this for instance meanse that for bold or italic typface, you must use the \mathbf{...} and \mathit{...} commands.

In addition to standard LaTeX commands, you can use the following IRIS-specific commands in the format strings:

- \sprintf{FFFF} - to modify the way each numeric entry that passes the test is printed by the sprintf function; FFFF is one of the standard sprintf formatting strings.
- \hide{?} - to hide the actual entry when it is supposed to be replaced with something else.

You can combine multiple tests and their correponding formats in one structure; they will be all applied to each entry in the specified order.

LHS-RHS plots

The LHS-RHS report graphs are still an experimental feature.

When the option `'yAxis='` is used to plot on both the LHS and the RHS y-axis, the plot functions are restricted to `@plot`, `@bar`, `@area` and `@stem`. Also, because of a bug in Matlab, always control the color of the lines, bars and areas in all LHS-RHS graphs: use either the option `'plotOptions='` in this command, or `'style='` in the respective graph P440 command.

Example (Conditional format structure)

Typeset negative values in italic, and values in periods before 2010Q1 blue:

```
cf = struct();
cf(1).test = 'value < 0';
cf(1).format = '\mathit{?}';
cf(2).test = 'date < qq(2010,1)';
cf(2).format = '\color{blue}';
```

---

subheading

Enter subheading in table

Syntax

```
P.subheading(CAP,...)
```

Input arguments

- `P` [struct] - Report object created by the `report.new` P450 function.
- `CAP` [char] - Text displayed as a subheading on a separate line in the table.

Options

- `'justify='` [`'c'` | `'l'` | `'r'`] - (Inheritable from parent objects) Horizontal alignment of the subheading (centre, left, right).

- 'separator=' [char | empty] - (Not inheritable from parent objects) LaTeX commands that will be inserted immediately after the end of the table row, i.e. appended to \, within a tabular mode.
- 'stretch=' [true | false] - (Inheritable from parent objects) Stretch the subheading text also across the data part of the table; if not the text will be contained within the initial descriptive columns.
- 'typeface=' [char | '\itshape\bfseries'] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the subheading; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

Generic options

See help on generic options P426 in report objects.

Description

Example

---

table

Start new table

Syntax

```
P.table(Caption,...)
```

Input arguments

- P [report] - Report object created by the report.new P450 function.
- Caption [char | cellstr] - Title or a cell array with title and subtitle displayed at the top of the table; see Description for splitting the title or subtitle into multiple lines.

Options

- 'arrayStretch=' [numeric | 1.15] - (Inheritable from parent objects) Stretch between lines in the table (in pts).
- 'captionTypeface=' [cell | '\large\bfseries'] - LaTeX format commands for typesetting the table caption and subcaption; you can use Inf for either to indicate the default format.
- 'colFootnote=' [cell | empty] - Footnotes for individual dates in the headings of the columns, or column names in user-defined tables; the option must be a cell array with date-footnote pairs.
- 'colHighlight=' [numeric | empty] - Dates for which the entire corresponding columns will be highlighted; because of a bug in the LaTex package colortbl, this option cannot be combined with the option 'rowHighlight=' in report/series P454.
- 'colStruct=' [struct | empty] - User-defined structure of the table columns; use of this option disables 'range='.
- 'colWidth=' [numeric | NaN] - (Inheritable from parent objects) Width, or a vector of widhts, of the table columns in emunits; NaN means the width of the column will adjust automatically.
- 'headlineJust=' ['c' | 'l' | 'r'] - Horizontal justification of the headline entries (individual dates or user-defined text): centre, left, right.
- 'footnote=' [char | empty] - Footnote at the table title; only shows if the title is non-empty.
- 'long=' [true | false] - (Inheritable from parent objects) If true, the table may stretch over more than one page.
- 'longFoot=' [char | empty] - (Inheritable from parent objects) Works only with 'long='=true: Footnote that appears at the bottom of the table (if it is longer than one page) on each page except the last one.
- 'longFootPosition=' ['centre' | 'left' | 'right'] - (Inheritable from parent objects) Works only with 'long=' true: Horizontal alignment of the footnote in long tables.
- 'range=' [numeric | empty] - (Inheritable from parent objects) Date range or vector of dates that will appear as columns of the table.
- 'separator=' [char | '\medskip\par'] - (Inheritable from parent objects) LaTeX commands that will be inserted after the table.
- 'sideways=' [true | false] - (Inheritable from parent objects) Print the table rotated by 90 degrees.
- 'tabcolsep=' [NaN | numeric] - (Inheritable from parent objects) Space between columns in the table, measured in em units; NaN means the LaTeX default.

- 'typeface=' [char | empty] - LaTeX code specifying the typeface for the table as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).
- 'vline=' [numeric | empty] - (Inheritable from parent objects) Vector of dates after which a vertical line (divider) will be placed.

Date format options

See dat2str P280 for details on date format options.

- 'dateFormat=' [char | cellstr | 'YYYYFP'] - Date format string, or array of format strings (possibly different for each date).
- 'freqLetters=' [char | 'YHQBMW'] - Six letters used to represent the six possible frequencies of IRIS dates, in this order: yearly, half-yearly, quarterly, bi-monthly, monthly, and weekly (such as the 'Q' in '2010Q1').
- 'months=' [cellstr | {'January',...,'December'}] - Twelve strings representing the names of the twelve months.
- 'standinMonth=' [numeric | 'last' | 1] - Month that will represent a lower-than-monthly-frequency date if the month is part of the date format string.

Generic options

See help on generic options P426 in report objects.

Description

Tables are top-level report objects and cannot be nested within other report objects, except align P427. Table objects can have the following children:

- series P454;
- subheading P458.

By default, the date row is printed as a leading row with dates formated using the option 'dateFormat='. Alternatively, you can specify this option as a cell array of two strings. In that case, the dates will be printed in two rows. The first row will have a date string displayed and centred for every year, and the first cell of the 'dateFormat=' option will be used for formatting. The second row will have a date displayed for every period (i.e. every column), and the second cell of the 'dateFormat=' option will be used for formatting.

User-defined structure of the table columns

Use the option 'colStruct=' to define your own table columns. This gives you more flexibility than when using the 'range=' option in defining the content of the table.

The option 'colStruct=' must be a 1-by-N struct, where N is the number of columns you want in the table, with the following fields:

- 'name=' - specifies the descriptor of the column that will be displayed in the headline;
- 'func=' - specifies a function that will be applied to the input series; if 'func=' is empty, no function will be applied. The function must evaluate to a tseries or a numeric scalar.
- 'date=' - specifies the date at which a number will be taken from the series unless the function 'func=' applied before resulted in a numeric scalar.

Titles and subtitles

The input argument Caption can be either a text string, or a 1-by-2 cell array of strings. In the latter case, the first cell will be printed as a title, and the second cell will be printed as a subtitle.

To split the title or subtitle into multiple lines, use the following LaTeX commands wrapped in curly brackets: {\\} or {\\[Xpt]}, where X is the width of an extra vertical space (in points) added between the respective lines.

Example

Compare the headers of these two tables:

```
x = report.new();

x.table('First table', ...
    'range',qq(2010,1):qq(2012,4), ...
    'dateformat','YYYYFP');
% You can add series or subheadings here.

x.table('Second table', ...
    'range,qq(2010,1):qq(2012,4), ...
    'dateformat',{'YYYY','FP'});
% You can add series or subheadings here.

x.publish('myreport.pdf');
```

tex

Include LaTeX code or verbatim input in report

Syntax with input specified in comment block

```
P.tex(Cap,...)

%{
Write text or \LaTeX\ code as a block comment
right after the P.tex( ) command.
%}
```

Syntax with input specified as char argument

```
P.tex(Cap,Code,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Cap [char] - Caption displayed at the top of the text.
- Code [char] - LaTeX code or text input that will be included in the report.

Options

- 'centering=' [true | false] - (Inheritable from parent objects) Centre the LaTeX code or text input on the page.
- 'footnote=' [char | empty] - Footnote at the tex block title; only shows if the title is non-empty.
- 'separator=' [char | '\medskip\par'] - (Inheritable from parent objects) LaTeX commands that will be inserted after the text.
- 'verbatim=' [true | false] - If true the text will be typeset verbatim in monospaced font; if false the text will be treated as LaTeX
code included in the report.

Generic options

See help on generic options P426 in report objects.

Description

Example

---

userfigure

Insert existing figure window

Syntax

```
P.userfigure(Caption,H,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char | cellstr] - Title or a cell array with title and subtitle displayed at the top of the figure; see Description for splitting the title or subtitle into multiple lines.
- H [numeric] - Handle to a graphics figure created by the user that will be captured and inserted in the report.

Options

See help on report/figure P437 for options available.

Generic options

See help on generic options P426 in report objects.

Description

The function report/userfigure inserts an existing figure window (created by the user by standard Matlab commands, and referenced by its handle, H) into a report:

- The figure and the graphs in it must be created before you call report/figure: any changes or additions to the figure or its graphs made after you call the function will not show in the report.
- The userfigure cannot have any children; in other words, you cannot call report/graph P440 after a call to report/userfigure.

Titles and subtitles

The input argument Caption can be either a text string, or a 1-by-2 cell array of strings. In the latter case, the first cell will be printed as a title, and the second cell will be printed as a subtitle.

To split the title or subtitle into multiple lines, use the following LaTeX commands wrapped in curly brackets: {\\} or {\\[Xpt]}, where X is the width of an extra vertical space (in points) added between the respective lines.

Figure window and figure handle

The figure H is saved to a fig file and stored within the report object. At the time of publishing the report, the figure is re-created again in a new separate window.

If the option 'close=' is set to false this new figure window will remain open after the report is published. The handle to this figure window will be included in the field .figureHandle of the information struct Info returned by report/publish P451.

Example

---

vline

Add vertical line to graph

Syntax

```
P.vline(Caption,Date,...)
```

Input arguments

- P [struct] - Report object created by the report.new P450 function.
- Caption [char] - Caption used to annotate the vertical line.
- Date [numeric] - Date at which the vertical line will be plotted.

Options

- 'hPosition=' ['bottom' | 'middle' | 'top'] - (Inheritable from parent objects) Horizontal position of the caption.
- 'vPosition=' ['centre' | 'left' | 'right'] - (Inheritable from parent objects) Vertical position of the caption relative to the line.
- 'timePosition=' ['after' | 'before' | 'middle'] - Placement of the vertical line on the time axis: in the middle of the specified period, immediately before it (between the specified period and the previous one), or immediately after it (between the specified period and the next one).

Generic options

See help on generic options P426 in report objects.

Description

Example

22 Quick Database Plots

Quick database plot functions

- dbplot P408 - Plot from database.

Getting on-line help on quick database plot functions

```
help dbase/dbplot
```

---

dbplot

Plot from database

Syntax

```
[FF,AA,PDb] = dbplot(D,List,Range,...)
[FF,AA,PDb] = dbplot(D,Range,List,...)
[FF,AA,PDb] = dbplot(D,List,...)
[FF,AA,PDb] = dbplot(D,Range,...)
[FF,AA,PDb] = dbplot(D,...)
```

Input arguments

- D [struct] - Database with input data.
- List [cellstr | rexp] - List of expressions (or labelled expressions) that will be evaluated and plotted in separate graphs; if not specified, all time series name found in the input database D will be plotted. Alternatively, List can be a regular expression (rexp object), which will be matched against all time series names in the input database.
- Range [numeric] - Date range; if not specified, the function dbrange P413 will be used to determined the plotted range (same for all graphs).

Output arguments

- FF [numeric] - Handles to figures created by qplot.
- AA [cell] - Handles to axes created by qplot.

- PDB [struct] - Database with actually plotted series.

Options

- 'addClick=' [true | false] - Make axes expand in a new graphics figure upon mouse click.
- 'captions=' [cellstr | @comment | *empty*] - Strings that will be used for titles in the graphs that have no title in the q-file.
- 'clear=' [numeric | empty] - Serial numbers of graphs (axes objects) that will not be displayed.
- 'dbSave=' [cellstr | empty] - Options passed to dbsave when 'saveAs=' is used.
- 'deviationsFrom=' [numeric | empty] - Each expression in List that starts with a @ or # (see Description) will be reported in deviations from this specified date.
- 'deviationsTimes=' [numeric | empty] - Used only if 'deviationsFrom=' is non-empty; each expression in List that starts with a @ or # (see Description) will be reported in deviations multiplied by this number.
- 'drawNow=' [true | false] - Call Matlab drawnow function upon completion of all figures.
- 'grid=' [true | false] - Add grid lines to all graphs.
- 'highlight=' [numeric | cell | empty] - Date range or ranges that will be highlighted.
- 'interpreter=' ['latex' | 'none'] - Interpreter used in graph titles.
- 'mark=' [cellstr | empty] - Marks that will be added to each legend entry to distinguish individual columns of multivariated tseries objects plotted.
- 'maxPerFigure=' [numeric | 36] - Maximum number of graphs in one figure window; if the actual graph count exceeds maxPerFigure, the option 'subplot=' is adjusted automatically, and new figure windows are opened as needed.
- 'overflow=' [true | false] - Open automatically a new figure window if the number of subplots exceeds the available total; 'overflow='false means an error will occur instead.
- 'plotFunc=' [@bar | @hist | *@plot* | @plotcmp | @plotpred | @stem | cell] - Plot function used to create the graphs; use a cell array, {plotFunc,...} to specify extra input arguments that will be passed into the plotting function.
- 'prefix=' [char | 'P%g_'] - Prefix (a sprintf format string) that will be used to precede the name of each entry in the PDb database.
- 'round=' [numeric | Inf] - Round the input data to this number of decimals before plotting.

- 'saveAs=' [char | empty] - File name under which the plotted data will be saved either in a CSV data file or a PS graphics file; you can use the 'dbsave=' option to control the options used when saving CSV.
- 'style=' [struct | empty] - Style structure that will be applied to all figures and their children created by the qplot function.
- 'subplot=' ['auto' | numeric] - Default subplot division of figures, can be modified in the q-file.
- 'sstate=' [struct | model | empty] - Database or model object from which the steady-state values referenced to in the quick-report file will be taken.
- 'style=' [struct | empty] - Style structure that will be applied to all created figures upon completion.
- 'transform=' [function_handle | empty] - Function that will be used to trans
- 'tight=' [true | false] - Make the y-axis in each graph tight.
- 'vLine=' [numeric | empty] - Dates at which vertical lines will be plotted.
- 'zeroLine=' [true | false] - Add a horizontal zero line to graphs whose y-axis includes zero.

Description

The function dbplot opens a new figure window (as many as needed to accommodate all graphs given the option 'subplot='), and creates a graph for each entry in the cell array List.

List can contain either the names of database fields, or expressions referring to database fields; these expressions will be then evaluated in the input database context. You can also add labels (that will be displayed as graph titles) enclosed in double quotes and preceding the expressions. Alternatively, you can specify titles through the option 'captions='. At the beginning of the expression, you can use one of the following marks:

- ^ (a hat symbol) means the function specified in the option 'transform=' will not be applied to that expression;
- @ (an at symbol) in combination with the option 'deviationFrom=' means that the deviations will reported in multiplicative form (i.e. the actual value divided by the base period value).
- # (a hash symbol) in combination with the option 'deviationFrom=' means that the deviations will reported in additive form (i.e. the actual value minus the base period value).

Example

The following command will plot the time series x and y as deviations from 1 multiplied by 100 (see the option 'transform='), and the time series z as it is (because of the ^ symbol at the beginning). The first series will be labeled simply 'x', while the last two series will be labeled 'Series y' and 'Series z', respectively.

```
dbplot(d,qq(2010,1):qq(2015,4), ...
   { 'x', '"Series y" y', '^"Series z"' }, ...
   'transform=',@(x) 100*(x-1));
```

Example

The following command will plot the time series x and y as deviations from year 2000; x will be computed as additive deviations (i.e. the base period value will be subtracted from its observations) whereas y will be computed as a multiplicative deviations (i.e. the observations will be divided by the base period value). The last time series z will not be transforme.d

```
dbplot(d,yy(2000):yy(2010), ...
   { '# x', '@ y', 'z' }, ...
   'deviationsFrom=',yy(2000));
```

Example

The following command will plot all time series found in the database that start with 'a'.

```
dbplot(d,rexp('^a.*'));
```

Example

Create an example database with the following fields: c, ctrend, y, ytrend, k, ktrend (the exact way these series are created is, of course, irrelevant):

```
range = qq(2000,1):qq(2004,4);
s = struct();
s.c = 1+cumsum( tseries(range,@rand)/10 );
s.ctrend = hpf(s.c);
s.y = 1+cumsum( tseries(range,@rand)/10 );
```

```
s.ytrend = hpf(s.y);
s.k = 1+ cumsum( tseries(range,@rand)/10 );
s.ktrend = hpf(s.k);
disp(s);
```

Plot the individual series against their respective trends, each in its own graph:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' } );
```

To automate this task, create the list of expressions to be plotted using the standard Matlab function `strcat`:

```
list = {'c','y','k'};
plotList = strcat( '[' , list , ',' , list , 'trend]' );
disp(plotList);
dbplot(s,range,plotList);
```

In the case of some complex transformation(s), e.g.

```
dbplot(s,range, { ...
    '100*log([c,ctrend])',  ...
    '100*log([y,ytrend])', ...
    '100*log([k,ktrend])' } );
```

use the option `'transform='` to apply the specified function to all series before they get plotted:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' }, ...
    'transform=',@(x) 100*log(x) );
```

If some graphs need to be excluded from `'transform='`, use a hat ^ at the beginning of the expression:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '^[k,ktrend]' }, ...
    'transform=',@(x) 100*log(x) );
```

Include titles for the individual graphs in double quotes at the beginning of each expression:

```
dbplot(s,range, { ...
    '"Consumption" [c,ctrend]', ...
    '"Output" [y,ytrend]', ...
    '"Capital" [k,ktrend]' } );
```

or alternatively use the option 'captions=' to do the same thing:

```
dbplot(s,range, ...
    { '[c,ctrend]', '[y,ytrend]', '[k,ktrend]' }, ....
    'captions=',{'Consumption','Output','Capital'} );
```

23 Graphics Functions (grfun Package)

Graphics functions

Getting on-line help on graphics functions

```
help grfun
help grfun/function_name
```

---

bottomlegend

Horizontal graph legend displayed at the bottom of the figure window

Syntax

```
Le = grfun.bottomlegend(Entry,Entry,...)
```

Input arguments

- Entry [char | cellstr] - Legend entries; same as in the standard legend function.

Output arguments

- AX [numeric] - Handle to the legend axes object created.

Description

Example

■ ftitle

Add title to figure window

Syntax

```
Aa = grfun.ftitle(Titles,...)
Aa = grfun.ftitle(FF,Titles,...)
```

Input arguments

- FF [numeric | struct] - Handle to a figure window or windows; or a struct that includes a field name figure.
- Titles [cellstr | char] - Text string to be centred, or cell array of strings to be placed on the LHS, centred, and on the RHS of the figure.

Output arguments

- Aa [numeric] - Handle or handles to annotation objects.

Options

- 'location=' ['north' | 'west' | 'east' | 'south'] - Location of the figure title: top, left edge sideways, right edge sideways, bottom.

Description

Example

■ highlight

Highlight specified range or date range in a graph

Syntax

```
[Pt,Cp] = highlight(Range,...)
[Pt,Cp] = highlight(Ax,Range,...)
```

Input arguments

- Range [numeric] - X-axis range or date range that will be highlighted.
- Ax [numeric] - Handle(s) to axes object(s) in which the highlight will be made.

Output arguments

- Pt [numeric] - Handle to the highlighted area (patch object).
- Cp [numeric] - Handle to the caption (text object).

Options

- 'caption=' [char] - Annotate the highlighted area with a text string.
- 'color=' [numeric | 0.8] - An RGB color code, a Matlab color name, or a scalar shade of gray.
- 'excludeFromLegend=' [true | false] - Exclude the highlighted area from legend.
- 'hPosition=' ['center' | 'left' | 'right'] - Horizontal position of the caption.
- 'vPosition=' ['bottom' | 'middle' | 'top' | numeric] - Vertical position of the caption.

Description

Example

---

hline

Add horizontal line with text caption at the specified position

Syntax

```
Ln = hline(Pos,...)
Ln = hline(Ax,Pos,...)
```

Input arguments

- 'Pos' [numeric] - Vertical position or vector of positions at which the horizontal line(s) will be drawn.
- Ax [numeric] - Handle to an axes object (graph) or to a figure window in which the the horizontal line will be added; if not specified the line will be added to the current axes.

Output arguments

- Ln [numeric] - Handle to the line plotted (line object).

Options

- 'excludeFromLegend=' [true | false] - Exclude the line from legend.

Any options valid for the standard plot function.

Description

Example

---

maxfigure

Maximize figure window

Syntax

```
Fig = maxfigure(H,...)
Fig = maxfigure(...)
```

Input arguments

- H [handle] - Handle to existing figure window that will be maximized; if omitted, a new maximized figure window will be created.

Output arguments

- Fig [numeric] - Handle to the figure created.

Options

See help on standar figure for the options available.

Description

The function maxfigure uses get(0,'screenSize') to determine the size of the screen, and sets the figure property 'outerPosition' accordingly.

Example

---

movetobkg

Move graphics objects to the background

Syntax

```
grfun.movetobkg(Parent,ToBkg)
```

Input arguments

- Parent [numeric] - Graphics handle to a parent object.
- ToBkg [numeric] - Graphics handle to children that will be moved to the background.

Description

Example

---

movetosubplot

Move an existing axes object or legend to specified subplot position

Syntax

```
Ax = grfun.movetosubplot(Ax,M,N,P)
Ax = grfun.movetosubplot(Ax,'bottom')
Ax = grfun.movetosubplot(Ax,'top')
```

Input arguments

- `Ax` [numeric] - Handle to an existing axes object or legend.
- `M`, `N`, `P` [numeric] - Specification of the new position; see help on standard `subplot`.

Output arguments

- `AX` [numeric] - Handle to the axes or legend moved to the new position.

Description

The syntax with `'bottom'` and `'top'` places the axes centered at, respectively, the bottom or top of the figure window.

Example

---

plotcircle

Draw a circle or disc

Syntax

```
H = grfun.plotcircle(X,Y,RAD,...)
```

Input arguments

- X [numeric] - X-axis location of the centre of the circle.
- Y [numeric] - Y-axis location of the centre of the circle.
- RAD [numeric] - Radius of the circle.

Output arguments

- H [numeric] - Handle to the line or the filled area.

Options

- 'fill=' [true | false] - Switch between a circle ('fill=' false) and a disc ('fill=' true).

Any property name-value pair valid for line graphs.

Description

Example

---

plotmat

Visualise 2D matrix

Syntax

```
[HPos,HNeg,HNanInf,HMax] = grfun.plotmat(X,...)
[HPos,HNeg,HNanInf,HMax] = plotmat(X,...)
```

Input arguments

- X [numeric] - 2D matrix that will be visualised; ND matrices will be unfolded in 2nd dimension before plotting.

Output arguments

- HPos [numeric] - Handles to discs displaying non-negative entries.
- HNeg [numeric] - Handles to discs displeying negative entries.
- HNanInf [numeric] - Handles to NaN or Inf marks.
- HMax [numeric] - Handles to circles displaying maximum value.

Options

- 'colNames=' [char | cellstr | empty | 'auto'] - Names that will be given to the columns of the matrix.
- 'rowNames=' [char | cellstr | empty | 'auto'] - Names that will be give to the row of the matrix.
- 'maxCircle=' [true | false] - If true,display a circle denoting the maximum value around each entry.
- 'nanInf=' [char | X] - Appearance of NaN and Inf entries.
- 'showDiag=' [true | false] - If false, hide the entries on the main diagonal by setting them to NaN.
- 'scale=' [numeric | 'auto'] - Maximum value (positive) relative to which all matrix entries will be scaled; by default the scale is the maximum entry in the input matrix, max(max(abs(X(isfinite(X))))).

Description

Example

---

plotneigh

Plot local behaviour of objective function after estimation

Syntax

```
H = grfun.plotneigh(D,...)
```

Input arguments

- D [struct] - Structure describing the local behaviour of the objective function returned by the neighbourhood P125 function.

Output arguments

- H [struct] - Struct with handles to the graphics objects plotted by plotpp; the struct has the following fields with vectors of handles: figure, axes, obj, est, lik, bounds.

Options

- 'caption=' [empty | cellstr] - User-supplied graph titles; if empty, default captions will be automatically created.
- 'model=' [model | empty] - Model object used to create graph captions if the option 'caption=' is 'descript' or 'alias'.
- 'plotObj=' [true | false] - Plot the local behaviour of the overall objective function; a cell array can be specified to control graphics options.
- 'plotLik=' [true | false | cell] - Plot the local behaviour of the data likelihood component; a cell array can be specified to control graphics options.
- 'plotEst=' [true | false | cell] - Mark the actual parameter estimate; a cell array can be specified to control graphics options.
- 'plotBounds=' [true | false | cell] - Draw the lower and/or upper bounds if they fall within the graph range; a cell array can be specified to control graphics options.
- 'subplot=' ['auto' | numeric] - Subplot division of the figure when plotting the results.
- 'title=' [{'interpreter','none'} | cell] - Display graph titles, and specify graphics options for the titles.
- 'linkAxes=' [true | false] - Make the vertical axes identical for all graphs.

Description

The data log-likelihood curves are shifted up or down by an arbitrary constant to make them fit in the graph; their curvature is preserved.

Example

plotpp

Plot prior and/or posterior distributions and/or posterior mode

Syntax

```
[PrG,PoG,H] = grfun.plotpp(E,[],[],...)
[PrG,PoG,H] = grfun.plotpp(E,Est,[],...)
[PrG,PoG,H] = grfun.plotpp(E,[],Theta,...)
[PrG,PoG,H] = grfun.plotpp(E,[],Stats,...)
[PrG,PoG,H] = grfun.plotpp(E,Est,Theta,...)
[PrG,PoG,H] = grfun.plotpp(E,Est,Stats,...)
```

Input arguments

- `E` [struct] - Estimation input struct, see `estimate` P85, with prior function handles from the logdist P196 package.
- `Est` [struct | empty] - Output struct returned by the `model/estimate` P85 function; `Est` will be used to plot the maximised posterior modes.
- `Theta` [numeric | empty] - Array with the chain of draws from the posterior simulator `arwm` P188.
- `Stats` [struct | empty] - Output struct returned by the posterior simulator statistics function `stats` P193.

Output arguments

- `PrG` [struct] - Struct with x- and y-axis coordinates to plot the prior distribution for each parameter.
- `PoG` [struct] - Struct with x- and y-axis coordinates to plot the posterior distribution for each parameter.
- `H` [struct] - Struct with handles to the graphics objects plotted by `plotpp`; the struct has the following fields with vectors of handles: `figure`, `axes`, `prior`, `poster`, `bounds`, `init`, `mode`, `title`.

Options

- `'caption='` [empty | cellstr] - User-supplied graph titles; if empty, default captions will be automatically created.

- 'describe=' ['auto' | true | false] - Include information on prior distributions, starting values, and maximised posterior modes in the graph titles; 'auto' means the descriptions will be shown only if 'plotPrior=' is true.
- 'ksdensity=' [numeric | empty] - Number of points over which the density will be calculated; if empty, default number will be used depending on the backend function available.
- 'plotInit=' [true | false | cell] - Plot starting values (initial consition used in posterior mode maximisation) as vertical stems.
- 'plotPrior=' [true | false | cell] - Plot prior distributions.
- 'plotMode=' [true | false | cell] - Plot maximised posterior modes as vertical stems; the modes are taken from Est (and not from Stats or Theta).
- 'plotPoster=' [true | false | cell] - Plot posterior distributions.
- 'plotBounds=' [true | false | cell] - Plot lower and/or upper bounds as vertical lines; if false, the bounds will be plotted only added if within the graph x-limits.
- 'sigma=' [numeric | 3] - Number of std devs from the mean or the mode (whichever covers a larger area) to the left and to right that will be plotted unless running out of bounds.
- 'tight=' [true | false] - Make graph axes tight.
- 'title=' [true | false | cell] - Display graph titles, and specify graphics options for the titles.
- 'xLims=' [struct | empty] - Control the x-limits of the prior and posterior graphs.

Description

The options that control what will be plotted in the graphs (i.e. 'plotInit=', 'plotPrior=', 'plotMode=', 'plotPoster=', 'plotBounds=','title=') can be set to one of the following three values:

- true,
- false,
- a cell array with sub-options to control the appearance of the respetive line; these will be passed into the respective plotting function.

Example

---

style

Apply styles to graphics object and its descandants

Syntax

```
grfun.style(H,S,...)
```

Input arguments

- H [numeric] - Handle to a figure or axes object that will be styled together with its descandants (unless 'cascade=' is false).
- S [struct] - Struct each field of which refers to an object-dot-property; the value of the field will be applied to the the respective property of the respective object; see below the list of graphics objects allowed.

Options

- 'cascade=' [true | false] - Cascade through all descendants of the object H; if false only the object H itself will be styled.
- 'warning=' [true | false] - Display warnings produced by this function.

Description

The style structure, S, is constructed of any number of nested object-property fields:

```
S.object.property = value;
```

The following is the list of standard Matlab grahics objects the top-level fields can refer to:

- figure
- axes
- title
- xlabel
- ylabel
- zlabel
- line
- bar
- patch
- text

Special object names

In addition to standard Matlab graphics object names, you can also refer to the following special instances of objects created by IRIS functions:

- `rhsaxes` (an RHS axes object created by `plotyy`)
- `legend` (represented by an axes object);
- `plotpred` (line objects with prediction data created by `plotpred`);
- `highlight` (a patch object created by `highlight`);
- `highlightcaption` (a text object created by `highlight`);
- `vline` (a patch object created by `vline`);
- `vlinecaption` (a text object created by `vline`);
- `zeroline` (a line object created by `zeroline`).

The property used as the second-level field is simply any regular Matlab property of the respective object (see Matlab help on graphics).

The value assigned to a particular property can be either of the following:

- a single proper valid value (i.e. a value you would be able to assign using the standard Matlab `set` function);
- a cell array of multiple different values that will be assigned to the objects of the same type in order of their creation;
- a text string starting with a double exclamation point, `!!`, followed by Matlab commands. The commands are expected to eventually create a variable named `SET` whose value will then assigned to the respective property. The commands have access to variable `H`, a handle to the current object.

Setting font size

Font size (in objects like axes, title, etc.) can be set to either a numeric scalar (which is the default Matlab behavior) or a character string describing a numerical value followed by a percent sign, such as `'150%'`. In that case, the font size will be set to the corresponding percentage of the current size.

Example

■ vline

Add vertical line with text caption at the specified position

Syntax

```
[Ln,Cp] = grfun.vline(Pos,...)
[Ln,Cp] = grfun.vline(Ax,Pos,...)
```

Input arguments

- Pos [numeric] - Horizontal position or vector of positions at which the vertical line(s) will be drawn.
- Ax [numeric] - Handle to an axes object (graph) or to a figure window in which the the line will be added; if not specified the line will be added to the current axes.

Output arguments

- Ln [numeric] - Handle to the vline(s) plotted (line objects).
- Cp [numeric] - Handle to the caption(s) created (text objects).

Options

- 'caption=' [char] - Annotate vline with a text string.
- 'excludeFromLegend=' [true | false] - Exclude vline from legend.
- 'hPosition=' ['center' | 'left' | 'right'] - Horizontal position of the caption.
- 'vPosition=' ['bottom' | 'middle' | 'top' | numeric] - Vertical position of the caption.
- 'timePosition=' ['after' | 'before' | 'middle'] - Placement of the vertical line on the time axis: in the middle of the specified period, immediately before it (between the specified period and the previous one), or immediately after it (between the specified period and the next one).

Description

Example

---

zeroline

Add zero line if Y-axis limits include zero

Syntax

```
Ln = zeroline(...)
Ln = zeroline(H,...)
```

Input arguments

- H [numeric] - Handle to an axes object (graph) or to a figure window in which the the line will be added; if not specified the line will be added to the current axes.

Output arguments

- Ln [numeric] - Handle to the line plotted (line object).

Options

Any options valid for the standard plot function.

Description

Example