ModelFlow

Ib Hansen

Jul 26, 2022

CONTENTS:

CHAPTER ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

TWO

INTRODUCTION

Dette er en prøve og en ande kllgg

CHAPTER

THREE

INSTALLATION

3.1 Install Miniconda

https://docs.conda.io/en/latest/miniconda.html to download the latest version 3.9

- open the file to start instalation
- asked to install for: select just me
- in the start menu: select anaconda prompt

3.2 Install Modelflow in the base enviroment

kdæoijpoijasdf

```
conda  install  -c ibh -c  conda-forge modelflow jupyter -y
pip install dash_interactive_graphviz
jupyter contrib nbextension install --user
jupyter nbextension enable hide_input_all/main
jupyter nbextension enable splitcell/splitcell
jupyter nbextension enable toc2/main
```

3.3 Install Modelflow in the separate enviroment

In this case we call the envioritement ‘mf’:

```
conda create -n mf -c ibh -c  conda-forge modelflow jupyter -y
conda activate mf
pip install dash_interactive_graphviz
jupyter contrib nbextension install --user
jupyter nbextension enable hide_input_all/main
jupyter nbextension enable splitcell/splitcell
jupyter nbextension enable toc2/main
```

3.4 In windows this can be useful

```
conda install xlwings
```

3.5 To update ModelFlow

```
conda update modelflow -c ibh -c conda-forge  -y
```

CHAPTER

FOUR

CORE MODULES, CREATES AND SOLVES MODEL INSTANCES

4.1 Modelclass, Defines the model class

Created on Mon Sep 02 19:41:11 2013

This module creates model class instances.

@author: Ib

`class modelclass.`**`node`**(*lev*, *parent*, *child*)

A named tuple used when to drawing the logical structure. Describes an edge of the dependency graph

Lev Level from start

Parent The parent

Child The child

`child`

Alias for field number 2

`lev`

Alias for field number 0

`parent`

Alias for field number 1

`class modelclass.`**`BaseModel`**(*i_eq=''*, *modelname='testmodel'*, *silent=False*, *straight=False*, *funks=[]*, *tabcomplete=True*, *previousbase=False*, *use_preorder=True*, *normalized=True*, *safeorder=False*, *var_description={}*, ***kwargs*)

Class which defines a model from equations

In itself the BaseModel is of no use.

The **model** class enriches BaseModel with additional Mixin classes which has additional methods and properties.

A model instance has a number of properties among which theese can be particular useful:

allvar Information regarding all variables

basedf A dataframe with first result created with this model instance

lastdf A dataframe with the last result created with this model instance

The two result dataframes are used for comparision and visualisation. The user can set both basedf and altdf.

`classmethod from_eq`(*equations*, *modelname='testmodel'*, *silent=False*, *straight=False*, *funks=[]*, *params={}*, *tabcomplete=True*, *previousbase=False*, *normalized=True*, *norm=True*, *sym=False*, *sep='\n'*, ***kwargs*)

Creates a model from macro Business logic language.

That is the model specification is first exploded.

Parameters

- `equations` – The model
- `modelname` – Name of the model. Defaults to 'testmodel'.
- `silent` – Suppress messages. Defaults to False.
- `straigth` – Don't reorder the model. Defaults to False.
- `funks` – Functions incorporated in the model specification . Defaults to [].
- `params` – For later use. Defaults to {}.
- `tabcomplete` – Allow tab compleetion in editor, for large model time consuming. Defaults to True.
- `previousbase` – Use previous run as basedf not the first. Defaults to False.
- `norm` – Normalize the model. Defaults to True.
- `sym` – If normalize do it symbolic. Defaults to False.
- `sep` – Seperate the equations. Defaults to newline.

Returns A model instance

`get_histmodel`()

return a model instance with a model which generates historic values for equations marked by a frml name I or IDENT

Uses *`find_hist_model`*

`analyzemodelnew`(*silent*)

Analyze a model

The function creats:**Self.allvar** is a dictory with an entry for every variable in the model the key is the variable name. For each endogeneous variable there is a directory with thees keys:

Maxlag The max lag for this variable

Maxlead The max Lead for this variable

Endo 1 if the variable is endogeneous (ie on the left hand side of =

Frml String with the formular for this variable

Frmlnumber The number of the formular

Varnr Number of this variable

Terms The frml for this variable translated to terms

Frmlname The frmlname for this variable

Startnr Start of this variable in gauss seidel solutio vector :Advanced:

Matrix This lhs element is a matrix

Dropfrml If this frml shoud be excluded from the evaluation.

In addition theese properties will be created:

Endogene Set of endogeneous variable in the model

Exogene Se exogeneous variable in the model

Maxnavlen The longest variable name

Blank An emty string which can contain the longest variable name

Solveorder The order in which the model is solved - initaly the order of the equations in the model

Normalized This model is normalized

Endogene_true Set of endogeneous variables in model if normalized, else the set of declared endogeneous variables

`smpl`(*start=''*, *slut=''*, *df=None*)

Defines the model.current_per which is used for calculation period/index when no parameters are issues the current current period is returned

Either none or all parameters have to be provided

`check_sim_smpl`(*databank*)

Checks if the current period (the SMPL) is can contain the lags and the leads

`set_smpl`(*start=''*, *slut=''*, *df=None*)

Sets the scope for the models time range, and restors it afterward

Parameters

- `start` – Start time. Defaults to ''.
- `slut` – End time. Defaults to ''.
- `df` (*Dataframe, optional*) – Used on a dataframe not self.basedf. Defaults to None.

`set_smpl_relative`(*start_ofset=0*, *slut_ofset=0*)

Sets the scope for the models time range relative to the current, and restores it afterward

`keepswitch`(*switch=False*, *scenarios='*'*)

temporary place basedf,lastdf in keep_solutions if scenarios contains * or ? they are separated by | else space

`property endograph`

Dependencygraph for currrent periode endogeneous variable, used for reorder the equations if self.safeorder is true feedback for all lags are included

safeorder was a fix to handle lags = -0 which unexpected was used in WB models. Now it is handeled in modelpattern

`property calculate_freq`

The number of operators in the model

`property flop_get`

The number of operators in the model prolog,core and epilog

`get_columnsnr`(*df*)

returns a dict a databanks variables as keys and column number as item used for fast getting and setting of variable values in the dataframe `outeval`(*databank*)

takes a list of terms and translates to a evaluater function called los

The model axcess the data through:Dataframe.value[rowindex+lag,coloumnindex] which is very efficient

`eqcolumns`(*a*, *b*)

compares two lists

`xgenr`(*databank*, *start=''*, *slut=''*, *silent=0*, *samedata=1*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

`findpos`()

find a startposition in the calculation array for a model places startposition for each variable in model.allvar[variable][‘startpos’] places the max startposition in model.maxstart

`make_gaussline`(*vx*, *nodamp=False*)

takes a list of terms and translates to a line in a gauss-seidel solver for simultanius models the variables are mapped to position in a vector which has all relevant varaibles lagged this is in order to provide opertunity to optimise data and solving

New version to take hand of several lhs variables. Dampning is not allowed for this. But can easely be implemented by makeing a function to multiply tupels

`make_resline`(*vx*)

takes a list of terms and translates to a line calculating line

`createstuff3`(*dfxx*)

Connect a dataframe with the solution vector used by the iterative sim2 solver) return a function to place data in solution vector and to retrieve it again.

`outsolve`(*order=''*, *exclude=[]*)

returns a string with a function which calculates a Gauss-Seidle iteration of a model exclude is list of endogeneous variables not to be solved uses: model.solveorder the order in which the variables is calculated model.allvar[v][“gauss”] the ccalculation

`make_solver`(*ljit=False*, *order=''*, *exclude=[]*, *cache=False*)

makes a function which performs a Gaus-Seidle iteration if ljit=True a Jittet function will also be created. The functions will be placed in: model.solve model.solve_jit

`base_sim`(*databank*, *start=''*, *slut=''*, *max_iterations=1*, *first_test=1*, *ljit=False*, *exclude=[]*, *silent=False*, *new=False*, *conv=[]*, *samedata=True*, *dumpvar=[]*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *lcython=False*, *setbase=False*, *setlast=True*, *alfa=0.2*, *sim=True*, *absconv=0.01*, *relconv=1e-05*, *debug=False*, *stats=False*, ***kwargs*)

solves a model with data from a databank if the model has a solve function else it will be created.

The default options are resonable for most use:

Parameters

- **`start,slut`** – Start and end of simulation, default as much as possible taking max lag into acount
- **`max_iterations`** – Max interations

- `first_test` – First iteration where convergence is tested
- `ljit` – If True Numba is used to compile just in time - takes time but speeds solving up
- `new` – Force creation a new version of the solver (for testing)
- `exclude` – Don't use use theese foormulas
- `silent` – Suppres solving informations
- `conv` – Variables on which to measure if convergence has been achived
- `samedata` – If False force a remap of datatrframe to solving vector (for testing)
- `dumpvar` – Variables to dump
- `ldumpvar` – toggels dumping of dumpvar
- `dumpwith` – with of dumps
- `dumpdecimal` – decimals in dumps
- `lcython` – Use Cython to compile the model (experimental)
- `alfa` – Dampning of formulas marked for dampning (<Z> in frml name)
- `sim` – For later use
- `absconv` – Treshold for applying relconv to test convergence
- `relconv` – Test for convergence
- `debug` – Output debug information
- `stats` – Output solving statistics

Return outdf A dataframe with the solution

`outres`(*order=''*, *exclude=[]*)

returns a string with a function which calculates a calculation for residual check exclude is list of endogeneous variables not to be solved uses: model.solveorder the order in which the variables is calculated

`make_res`(*order=''*, *exclude=[]*)

makes a function which performs a Gaus-Seidle iteration if ljit=True a Jittet function will also be created. The functions will be placed in:

- model.solve
- model.solve_jit

`base_res`(*databank*, *start=''*, *slut=''*, *silent=1*, ***kwargs*)

calculates a model with data from a databank Used for check wether each equation gives the same result as in the original databank'

`class modelclass.Org_model_Mixin`

The model class, used for calculating models

Compared to BaseModel it allows for simultaneous model and contains a number of properties and functions to analyze and manipulate models and visualize results.

`property lister`

lists used in the equations

Returns Dictionary of lists defined in the input equations.

Return type dict

`property listud`

returns a string of the models listdefinitions

used when ceating (small) models based on this model

`vlist`(*pat*)

Returns a list of variable in the model matching the pattern, the pattern can be a list of patterns

Parameters `pat` (*string or list of strings*) – One or more pattern seperated by space wildcards * and ?, special pattern: #ENDO

Returns list of variable names matching the pat.

Return type out (list)

`static list_names`(*input*, *pat*, *sort=True*)

returns a list of variable in input matching the pattern, the pattern can be a list of patterns

`exodif`(*a=None*, *b=None*)

Finds the differences between two dataframes in exogeneous variables for the model Defaults to getting the two dataframes (basedf and lastdf) internal to the model instance

Exogeneous with a name ending in <endo>__RES are not taken in, as they are part of a un_normalized model

Parameters

- `a` (*TYPE, optional*) – DESCRIPTION. Defaults to None. If None model.basedf will be used.
- `b` (*TYPE, optional*) – DESCRIPTION. Defaults to None. If None model.lastdf will be used.

Returns the difference between the models exogenous variables in a and b.

Return type DataFrame

`get_eq_values`(*varnavn*, *last=True*, *databank=None*, *nolag=False*, *per=None*, *showvar=False*, *alsoendo=False*)

Returns a dataframe with values from a frml determining a variable

options:

last the lastdf is used else baseline dataframe

nolag only line for each variable

`get_eq_dif`(*varnavn*, *filter=False*, *nolag=False*, *showvar=False*)

returns a dataframe with difference of values from formula

`get_var_growth`(*varname*, *showname=False*, *diff=False*)

Returns the growth rate of this variable in the base and the last dataframe

`get_values`(*v*, *pct=False*)

returns a dataframe with the data points for a node, including lags

`__getitem__`(*name*)

To execute the index operator []

Uses the `vis` operator

`todynare`(*paravars=[]*, *paravalues=[]*)

This is a function which converts a Modelflow model instance to Dynare .mod format class modelclass.**Model_help_Mixin**

Helpers to model

static **timer**(*input='test'*, *show=True*, *short=True*)

A timer context manager, implemented using a generator function. This one will report time even if an exception occurs the time in seconds can be retrieved by <retunr value>.seconds “””

Parameters

- **input** (*string, optional*) – a name. The default is ‘test’.
- **show** (*bool, optional*) – show the results. The default is True.
- **short** (*bool, optional*) – . The default is False.

Return type None.

static **update_old**(*indf*, *updates*, *lprint=False*, *scale=1.0*, *create=True*, *keep_growth=False*, *start=''*, *end=''*)

Updates a dataframe and returns a dataframe

Parameters

- **indf** (*DataFrame*) – input dataframe.
- **basis** (*string*) – lines with variable updates look below.
- **lprint** (*bool, optional*) – if True each update is printed Defaults to False.
- **scale** (*float, optional*) – A multiplier used on all update input . Defaults to 1.0.
- **create** (*bool, optional*) – Creates a variables if not in the dataframe . Defaults to True.
- **keep_growth** (*bool, optional*) – Keep the growth rate after the update time frame. Defaults to False.
- **start** (*string, optional*) – Global start
- **end** – Global end

static **update**(*indf*, *updates*, *lprint=False*, *scale=1.0*, *create=True*, *keep_growth=False*)

Updates a dataframe and returns a dataframe

Parameters

- **indf** (*DataFrame*) – input dataframe.
- **basis** (*string*) – lines with variable updates look below.
- **lprint** (*bool, optional*) – if True each update is printed Defaults to False.
- **scale** (*float, optional*) – A multiplier used on all update input . Defaults to 1.0.
- **create** (*bool, optional*) – Creates a variables if not in the dataframe . Defaults to True.
- **keep_growth** (*bool, optional*) – Keep the growth rate after the update time frame. Defaults to False.

Returns the updated dataframe .

Return type df (TYPE)

A line in updates looks like this:

´<[[start] end]>` <var> <=|+|*|%|=growth|+growth|=diff> <value>… [–keep_growth_rate|–no_keep_growth_rate]

insertModelVar(*dataframe*, *addmodel=[]*)

Inserts all variables from this model, not already in the dataframe. If model is specified, the dataframw will contain all variables from this and model models.

also located at the module level for backward compability

class **defsub**

A subclass of dict. if a *defsub* is indexed by a nonexisting keyword it just return the keyword

test_model(*base_input*, *start=None*, *end=None*, *maxvar=1000000*, *maxerr=100*, *tol=0.0001*, *showall=False*, *dec=8*, *width=30*, *ref_df=None*)

Compares a straight calculation with the input dataframe.

shows which variables dont have the same value

Very useful when implementing a model where the results are known

Parameters

- **df** (*DataFrame*) – dataframe to run.
- **start** (*index, optional*) – start period. Defaults to None.
- **end** (*index, optional*) – end period. Defaults to None.
- **maxvar** (*int, optional*) – how many variables are to be chekked. Defaults to 1_000_000.
- **maxerr** (*int, optional*) – how many errors to check Defaults to 100.
- **tol** (*float, optional*) – check for absolute value of difference. Defaults to 0.0001.
- **showall** (*bolean, optional*) – show more . Defaults to False.
- **ref_df** (*DataFrame, optional*) – this dataframe is used for reference, used if add_factors has been calculated

Returns None.

class modelclass.**Dekomp_Mixin**

This class defines methods and properties related to equation attribution analyses (dekomp)

dekomp(*varnavn*, *start=''*, *end=''*, *basedf=None*, *altdf=None*, *lprint=True*, *time_att=False*)

Print all variables that determines input variable (varnavn) optional – enter period and databank to get var values for chosen period

dekomp_plot_per(*varnavn*, *sort=False*, *pct=True*, *per=''*, *threshold=0.0*, *rename=True*, *time_att=False*, *ysize=7*)

Returns a waterfall diagram with attribution for a variable in one time frame

Parameters

- **varnavn** (*TYPE*) – variable name.
- **sort** (*TYPE, optional*) – . The default is False.
- **pct** (*TYPE, optional*) – display pct contribution . The default is True.
- **per** (*TYPE, optional*) – DESCRIPTION. The default is ‘’.

- **threshold** (*TYPE, optional*) – cutoff. The default is 0.0.
- **rename** (*TYPE, optional*) – Use descriptions instead of variable names. The default is True.
- **time_att** (*TYPE, optional*) – Do time attribution . The default is False.

Return type a matplotlib figure instance .

get_att_pct(*n*, *filter=False*, *lag=True*, *start=''*, *end=''*, *time_att=False*)

det attribution pct for a variable. I little effort to change from multiindex to single node name

get_att_pct_to_from(*to_var*, *from_var*, *lag=False*, *time_att=False*)

Get the attribution for a singel variable

get_att_level(*n*, *filter=False*, *lag=True*, *start=''*, *end=''*, *time_att=False*)

det attribution pct for a variable. I little effort to change from multiindex to single node name

dekomp_plot(*varnavn*, *sort=True*, *pct=True*, *per=''*, *top=0.9*, *threshold=0.0*, *lag=True*, *rename=True*, *time_att=False*)

Returns a chart with attribution for a variable over the smpl

Parameters

- **varnavn** (*TYPE*) – variable name.
- **sort** (*TYPE, optional*) – . The default is False.
- **pct** (*TYPE, optional*) – display pct contribution . The default is True.
- **per** (*TYPE, optional*) – DESCRIPTION. The default is ‘’.
- **threshold** (*TYPE, optional*) – cutoff. The default is 0.0.
- **rename** (*TYPE, optional*) – Use descriptions instead of variable names. The default is True.
- **time_att** (*TYPE, optional*) – Do time attribution . The default is False.
- **lag** (*TYPE, optional*) – separete by lags The default is True.
- **top** (*TYPE, optional*) – where to place the title

Return type a matplotlib figure instance .

get_dekom_gui(*var=''*)

Interactive wrapper around `dekomp_plot` and `dekomp_plot_per`

Parameters **var** (*TYPE, optional*) – start variable . Defaults to ‘’.

Returns dict of matplotlib figs .

Return type show (TYPE)

totexplain(*pat='*'*, *vtype='all'*, *stacked=True*, *kind='bar'*, *per=''*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*)

makes a total explanation for the variables defined by pat

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘*’.
- **vtype** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘all’.
- **stacked** (*TYPE, optional*) – DESCRIPTION. Defaults to True.

- **kind** (*TYPE, optional*) – DESCRIPTION. Defaults to 'bar'.
- **per** (*TYPE, optional*) – DESCRIPTION. Defaults to ''.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ''.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to 'level'.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.

Returns DESCRIPTION.

Return type fig (TYPE)

`get_att_gui`(*var='FY'*, *spat='*'*, *desdic={}*, *use='level'*, *ysize=7*)

Creates a jupyter ipywidget to display model level attributions

`class modelclass.Description_Mixin`

This Class defines description related methods and properties

`static html_replace`(*ind*)

Replace special characters in html

`var_des`(*var*)

Returns blank if no description

`get_eq_des`(*var*, *show_all=False*)

Returns a string of descriptions for all variables in an equation:

`static read_wb_xml_var_des`(*filename*)

Read a xml file with variable description world bank style

`static languages_wb_xml_var_des`(*filename*)

Find languages in a xml file with variable description world bank style

`set_wb_xml_var_description`(*filename*, *language='English'*)

set variable descriptions from a xml file with variable description world bank style

`enrich_var_description`(*var_description*)

Takes a dict of variable descriptions and enhance it for the standard suffixes for generated variables

`class modelclass.Modify_Mixin`

Class to modify a model with new equations, (later alse delete, and new normalization)

`eqflip`(*flip=None*, *calc_add=True*, *newname=''*, *sep='\n'*)

Parameters

- **newnormalisation** (*TYPE, optional*) – Not implementet yet . The default is None.
- **newfunks** (*TYPE, optional*) – Additional userspecified functions. The default is [].
- **calc_add** (*bool, optional*) – Additional userspecified functions. The default is [].

Returns

- **newmodel** (*TYPE*) – The new model with the new and deleted equations .
- **newdf** (*TYPE*) – a dataframe with calculated add factors. Origin is the original models lastdf.

`eqdelete`(*deleteeq=None*, *newname=''*)

Parameters `deleteeq` (*TYPE, optional*) – Variables where equations are to be deleted. The default is None.

Returns

- **newmodel** (*TYPE*) – The new model with the new and deleted equations .
- **newdf** (*TYPE*) – a dataframe with calculated add factors. Origin is the original models lastdf.

`equpdate`(*updateeq=None*, *newfunks=[]*, *add_add_factor=False*, *calc_add=True*, *do_preprocess=True*, *newname=''*)

Parameters

- `updateeq` (*TYPE*) – new equations seperated by newline .
- `newfunks` (*TYPE, optional*) – Additional userspecified functions. The default is [].
- `calc_add` (*bool, optional*) – Additional userspecified functions. The default is [].

Returns

- **newmodel** (*TYPE*) – The new model with the new and deleted equations .
- **newdf** (*TYPE*) – a dataframe with calculated add factors. Origin is the original models lastdf.

`equpdate_old`(*updateeq=None*, *newfunks=[]*, *add_adjust=False*, *calc_add=True*, *do_preprocess=True*, *newname=''*)

Parameters

- `updateeq` (*TYPE*) – new equations seperated by newline .
- `newfunks` (*TYPE, optional*) – Additional userspecified functions. The default is [].
- `calc_add` (*bool, optional*) – Additional userspecified functions. The default is [].

Returns

- **newmodel** (*TYPE*) – The new model with the new and deleted equations .
- **newdf** (*TYPE*) – a dataframe with calculated add factors. Origin is the original models lastdf.

`class modelclass.Graph_Mixin`

This class defines graph related methods and properties

`static create_strong_network`(*g*, *name='Network'*, *typeout=False*, *show=False*)

create a solveorder and blockordering of af graph uses networkx to find the core of the model

`property strongfrml`

To search simultaneity (circularity) in a model this function returns the equations in each strong block

`superblock()`

finds prolog, core and epilog variables

`property prevar`

returns a set with names of endogenopus variables which do not depend on current endogenous variables `property epivar`

returns a set with names of endogenopus variables which do not influence current endogenous variables

`property preorder`

the endogenous variables which can be calculated in advance

`property epiorder`

the endogenous variables which can be calculated in advance

`property coreorder`

the solution order of the endogenous variables in the simultaneous core of the model

`property coreset`

The set of variables of the endogenous variables in the simultaneous core of the model

`property totgraph_nolag`

The graph of all variables, lagged variables condensed

`property totgraph`

Returns the total graph of the model, including leads and lags

`property endograph_nolag`

Dependencygraph for all periode endogeneous variable, shows total dependencies

`property endograph_lag_lead`

Returns the graph of all endogeneous variables including lags and leads

totgraph_get(*onlyendo=False*)

The graph of all variables including and seperate lagged and leaded variable

onlyendo : only endogenous variables are part of the graph

graph_remove(*paralist*)

Removes a list of variables from the totgraph and totgraph_nolag mostly used to remove parmeters from the graph, makes it less crowded

graph_restore()

If nodes has been removed by the graph_remove, calling this function will restore them

`class modelclass.Graph_Draw_Mixin`

This class defines methods and properties which draws and vizualize using different graphs of the model

treewalk(*g*, *navn*, *level=0*, *parent='Start'*, *maxlevel=20*, *lpre=True*)

Traverse the call tree from name, and returns a generator

to get a list just write: list(treewalk(…)) maxlevel determins the number of generations to back up

lpre=0 we walk the dependent lpre=1 we walk the precednc nodes

drawendo(***kwargs*)

draws a graph of of the whole model

drawendo_lag_lead(***kwargs*)

draws a graph of of the whole model

drawmodel(*lag=True*, ***kwargs*)

draws a graph of of the whole model

`plotadjacency`(*size=(5, 5)*, *title='Structure'*, *nolag=False*)

Draws an adjacendy matrix

Parameters

- **size** (*TYPE, optional*) – DESCRIPTION. Defaults to (5, 5).
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to 'Structure'.
- **nolag** (*TYPE, optional*) – DESCRIPTION. Defaults to False.

Returns A adjacency matrix drawing.

Return type fig (matplotlib figure)

`draw`(*navn*, *down=1*, *up=1*, *lag=False*, *endo=False*, *filter=0*, ***kwargs*)

draws a graph of dependensies of navn up to maxlevel

Lag show the complete graph including lagged variables else only variables.

Endo Show only the graph for current endogenous variables

Down level downstream

Up level upstream

`trans`(*ind*, *root*, *transdic=None*, *debug=False*)

as there are many variable starting with SHOCK, the can renamed to save nodes

`upwalk`(*g*, *navn*, *level=0*, *parent='Start'*, *maxlevel=20*, *filter=0.0*, *lpre=True*)

Traverse the call tree from name, and returns a generator

to get a list just write: list(upwalk(…)) maxlevel determins the number of generations to back maxlevel

`upwalk_old`(*g*, *navn*, *level=0*, *parent='Start'*, *up=20*, *select=0.0*, *lpre=True*)

Traverse the call tree from name, and returns a generator

to get a list just write: list(upwalk(…)) up determins the number of generations to back up

`explain`(*var*, *up=1*, *start=''*, *end=''*, *filter=0*, *showatt=True*, *lag=True*, *debug=0*, *noshow=False*, *dot=False*, ***kwargs*)

Walks a tree to explain the difference between basedf and lastdf

Parameters: :var: the variable we are looking at :up: how far up the tree will we climb :select: Only show the nodes which contributes :showatt: Show the explanation in pct :lag: If true, show all lags, else aggregate lags for each variable. :HR: if true make horisontal graph :title: Title :saveas: Filename :pdf: open the pdf file :svg: display the svg file :browser: if true open the svg file in browser :noshow: Only return the resulting graph :dot: Return the dot file only

`todot`(*g*, *navn=''*, *browser=False*, ***kwargs*)

makes a drawing of subtree originating from navn all is the edges attributex can be shown

Sink variale to use as sink

Svg Display the svg image

`gdraw`(*g*, ***kwargs*)

draws a graph of of the whole model

`maketip`(*v*, *html=False*)

Return a tooltip for variable v.

For use when generating .dot files for Graphviz

If html==True it can be incorporated into html string

makedotnew(*alledges*, *navn=''*, ***kwargs*)

makes a drawing of all edges in list alledges all is the edges

this can handle both attribution and plain

All show values for .dfbase and .lastdf

Last show the values for .lastdf

Growthshow Show growthrates

Attshow Show attributiuons

Filter Prune tree branches where all(abs(attribution)<filter value)

Sink variale to use as sink

Source variale to use as ssource

Svg Display the svg image in browser

Pdf display the pdf result in acrobat reader

Saveas Save the drawing as name

Size figure size default (6,6)

Warnings warnings displayed in command console, default =False

Invisible set of invisible nodes

Labels dict of labels for edges

Transdic dict of translations for consolidation of nodes {'SHOCK[_A-Z]*__J':'SHOCK__J','DEV__[_A-Z]*':'DEV'}

Dec decimal places in numbers

HR horisontal orientation default = False

Des inject variable descriptions

Fokus Variable to get special colour

Fokus2 Variable for which values are shown

Fokus2all Show values for all variables

todot2(*alledges*, *navn=''*, ***kwargs*)

makes a drawing of all edges in list alledges all is the edges

All show values for .dfbase and .dflaste

Last show the values for .dflast

Sink variale to use as sink

Source variale to use as ssource

Svg Display the svg image in browser

Pdf display the pdf result in acrobat reader

Saveas Save the drawing as name

Size figure size default (6,6)

Warnings warnings displayed in command console, default =False

Invisible set of invisible nodes

Labels dict of labels for edges

Transdic dict of translations for consolidation of nodes {'SHOCK[_A-Z]*__J':'SHOCK__J','DEV__[_A-Z]*':'DEV'}

Dec decimal places in numbers

HR horisontal orientation default = False

Des inject variable descriptions

`static display_graph`(*out*, *fname*, ***kwargs*)

Generates a graphviz file from the commands in out.

The file is placed in cwd/graph

A png and a svg file is generated, and the svg file is displayed if possible.

options pdf and eps determins if a pdf and an eps file is genrated.

option fpdf will cause the graph displayed in a seperate pdf window

option browser determins if a seperate browser window is open

`class modelclass.Json_Mixin`

This mixin class can dump a model and solution as json serialiation to a file.

allows the precooking of a model and solution, so a user can use a model without specifying it in a session.

`modeldump`(*outfile=''*, *keep=False*)

Dumps a model and its lastdf to a json file

if keep=True the model.keep_solutions will alse be dumped

`classmethod modelload`(*infile*, *funks=[]*, *run=False*, *keep_json=False*, ***kwargs*)

Loads a model and an solution

`class modelclass.Excel_Mixin`

This Mixin handels dumps and loads models into excel

`modeldump_excel`(*file*, *fromfile='control.xlsm'*, *keep_open=False*)

Dump model and dataframe to excel workbook

Parameters

- **file** (*TYPE*) – filename.
- **keep_open** (*TYPE, optional*) – Keep the workbook open in excel after returning, The default is False.

Returns **wb** – xlwings instance of workbook .

Return type TYPE

`class modelclass.Zip_Mixin`

This experimental class zips a dumped file

`class modelclass.Solver_Mixin`

This Mixin handels the solving of models.

`__call__`(**args*, ***kwargs*)

Runs a model.

Default a straight model is calculated by *xgenr* a simultaneous model is solved by *sim*

Sim If False forces a model to be calculated (not solved) if True force simulation

Setbase If True, place the result in model.basedf

Setlast if False don't place the results in model.lastdf

if the modelproperty previousbase is true, the previous run is used as basedf.

`is_newdata`(*databank*)

Determins if thius is the same databank as in the previous solution

`sim`(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=5*, *max_iterations=200*, *conv='*'*, *absconv=0.01*, *relconv=1e-07*, *stringjit=True*, *transpile_reset=False*, *dumpvar='*'*, *init=False*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *ljit=False*, *timeon=False*, *fairopt={'fair_max_iterations ': 1}*, *progressbar=False*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function

Then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

Solves using Gauss-Seidle

Parameters

- **databank** (*dataframe*) – Input dataframe
- **start** (*optional*) – start of simulation, defaults to ''
- **slut** (*optional*) – end of simulation, defaults to ''
- **silent** (*bool, optional*) – keep simulation silent , defaults to 1
- **samedata** (*bool, optional*) – the inputdata has exactly same structure as last simulation , defaults to 0
- **alfa** (*float, optional*) – Dampeing factor, defaults to 1.0
- **stats** (*bool, optional*) – Show statistic after finish, defaults to False
- **first_test** (*int, optional*) – Start testing af number og simulation, defaults to 5
- **max_iterations** (*int, optional*) – Max iterations, defaults to 200
- **conv** (*str, optional*) – variables to test for convergence, defaults to '*'
- **absconv** (*float, optional*) – Test convergence for values above this, defaults to 0.01
- **relconv** (*float, optional*) – If relative movement is less, then convergence , defaults to DEFAULT_relconv
- **stringjit** (*bool, optional*) – If just in time compilation do it on a string not a file to import, defaults to True
- **transpile_reset** (*bool, optional*) – Ingnore previous transpiled model, defaults to False
- **dumpvar** (*str, optional*) – Variables for which to dump the iterations, defaults to '*'
- **init** (*bool, optional*) – If True take previous periods value as starting value, defaults to False
- **ldumpvar** (*bool, optional*) – Dump iterations, defaults to False

- dumpwith (*int, optional*) – DESCRIPTION, defaults to 15
- dumpdecimal (*int, optional*) – DESCRIPTION, defaults to 5
- chunk (*int, optional*) – Chunk size of transpiled model, defaults to 30
- ljit (*bool, optional*) – Use just in time compilation, defaults to False
- timeon (*bool, optional*) – Time the elements, defaults to False
- fairopt (*TYPE, optional*) – Fair taylor options, defaults to {'fair_max_iterations': 1}
- progressbar (*TYPE, optional*) – Show progress bar , defaults to False

Returns A dataframe wilt the results

static grouper(*iterable*, *n*, *fillvalue=''*)

Collect data into fixed-length chunks or blocks

outsolve2dcunk(*databank*, *debug=1*, *chunk=None*, *ljit=False*, *type='gauss'*, *cache=False*)

takes a list of terms and translates to a evaluater function called los

The model axcess the data through:Dataframe.value[rowindex+lag,coloumnindex] which is very efficient

sim1d(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=1*, *max_iterations=100*, *conv='*'*, *absconv=1.0*, *relconv=1e-07*, *init=False*, *dumpvar='*'*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *ljit=False*, *stringjit=True*, *transpile_reset=False*, *fairopt={'fair_max_iterations ': 1}*, *timeon=0*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

outsolve1dcunk(*debug=0*, *chunk=None*, *ljit=False*, *cache='False'*)

takes a list of terms and translates to a evaluater function called los

The model axcess the data through:Dataframe.value[rowindex+lag,coloumnindex] which is very efficient

newton(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=1*, *newton_absconv=0.001*, *max_iterations=20*, *conv='*'*, *absconv=1.0*, *relconv=1e-07*, *nonlin=False*, *timeit=False*, *newton_reset=1*, *dumpvar='*'*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *ljit=False*, *stringjit=True*, *transpile_reset=False*, *lnjit=False*, *init=False*, *newtonalfa=1.0*, *newtonnodamp=0*, *forcenum=True*, *fairopt={'fair_max_iterations ': 1}*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

newtonstack(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=1*, *newton_absconv=0.001*, *max_iterations=20*, *conv='*'*, *absconv=1.0*, *relconv=1e-07*, *dumpvar='*'*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *nchunk=30*, *ljit=False*, *stringjit=True*, *transpile_reset=False*, *nljit=0*, *fairopt={'fair_max_iterations ': 1}*, *debug=False*, *timeit=False*, *nonlin=False*, *newtonalfa=1.0*, *newtonnodamp=0*, *forcenum=True*, *newton_reset=False*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

`newton_un_normalized`(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=1*, *newton_absconv=0.001*, *max_iterations=20*, *conv='*'*, *absconv=1.0*, *relconv=1e-07*, *nonlin=False*, *timeit=False*, *newton_reset=1*, *dumpvar='*'*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *ljit=False*, *stringjit=True*, *transpile_reset=False*, *lnjit=False*, *fairopt={'fair_max_iterations ': 1}*, *newtonalfa=1.0*, *newtonnodamp=0*, *forcenum=True*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

`newtonstack_un_normalized`(*databank*, *start=''*, *slut=''*, *silent=1*, *samedata=0*, *alfa=1.0*, *stats=False*, *first_test=1*, *newton_absconv=0.001*, *max_iterations=20*, *conv='*'*, *absconv=1.0*, *relconv=1e-07*, *dumpvar='*'*, *ldumpvar=False*, *dumpwith=15*, *dumpdecimal=5*, *chunk=30*, *nchunk=None*, *ljit=False*, *nljit=0*, *stringjit=True*, *transpile_reset=False*, *fairopt={'fair_max_iterations ': 1}*, *debug=False*, *timeit=False*, *nonlin=False*, *newtonalfa=1.0*, *newtonnodamp=0*, *forcenum=True*, *newton_reset=False*, ***kwargs*)

Evaluates this model on a databank from start to slut (means end in Danish).

First it finds the values in the Dataframe, then creates the evaluater function through the *outeval* function (`modelclass.model.fouteval()`) then it evaluates the function and returns the values to a the Dataframe in the databank.

The text for the evaluater function is placed in the model property **make_los_text** where it can be inspected in case of problems.

`res`(*databank*, *start=''*, *slut=''*, *debug=False*, *timeit=False*, *silent=False*, *chunk=None*, *ljit=0*, *stringjit=True*, *transpile_reset=False*, *alfa=1*, *stats=0*, *samedata=False*, ***kwargs*)

calculates the result of a model, no iteration or interaction The text for the evaluater function is placed in the model property **make_res_text** where it can be inspected in case of problems.

`invert`(*databank*, *targets*, *instruments*, *silent=1*, *DefaultImpuls=0.01*, *defaultconv=0.001*, *nonlin=False*, *maxiter=30*, ***kwargs*)

Solves instruments for targets

`errfunk1d`(*a*, *linenr*, *overhead=4*, *overeq=0*)

Handle errors in sim1d

`errfunk`(*values*, *linenr*, *overhead=4*, *overeq=0*)

developement function

to handle run time errors in model calculations

show_iterations(*pat='*'*, *per=''*, *last=0*, *change=False*, *top=0.9*)

shows the last iterations for the most recent simulation. iterations are dumped if ldumpvar is set to True variables can be selceted by: dumpvar = '<pattern>'

Parameters

- **pat** (*TYPE, optional*) – Variables for which to show iterations . Defaults to '*'.
- **per** (*TYPE, optional*) – The time frame for which to show iterations, Defaults to the last projection .
- **last** (*TYPE, optional*) – Only show the last iterations . Defaults to 0.
- **change** (*TYPE, optional*) – show the changes from iteration to iterations instead of the levels. Defaults to False.
- **top** (*float,optional*) – top of chartss between 1 and 0

Returns DESCRIPTION.

Return type fig (TYPE)

class modelclass.**Dash_Mixin**

This mixin wraps call the Dash dashboard

class modelclass.**WB_Mixin**

This mixin handles a number of enhancements

property **wb_behavioral**

returns endogeneous where the frml name contains a Z which signals a stocastic equation

property **wb_ident**

returns endogeneous variables not in wb_behavioral

fix(*df*, *pat='*'*, *start=''*, *end=''*)

Fixes variables to the current values.

for variables where the equation looks like:

```
var = (rhs)*(1-var_d)+var_x*var_d
```

The values in the smpl set by *start* and *end* will be set to:

```
var_x = var
var_d = 1
```

The variables fulfilling this are elements of .wb_behavioral

Parameters

- **df** (*TYPE*) – Input dataframe should contain a solution and all variables ..
- **pat** (*TYPE, optional*) – Select variables to endogenize. Defaults to '*'.
- **start** (*TYPE, optional*) – start periode. Defaults to ''.
- **end** (*TYPE, optional*) – end periode. Defaults to ''.

Returns the resulting daaframe .

Return type dataframe (TYPE)

unfix(*df*, *pat='*'*, *start=''*, *end=''*)

Unfix (endogenize) variables

Parameters

- **df** (*Dataframe*) – Input dataframe, should contain a solution and all variables .
- **pat** (*string, optional*) – Select variables to endogenize. Defaults to '*'.
- **start** (*TYPE, optional*) – start periode. Defaults to ''.
- **end** (*TYPE, optional*) – end periode. Defaults to ''.

Returns A dratframe with all dummies for the selected variablse set to 0 .

Return type dataframe (TYPE)

find_fix_dummy_fixed(*df=None*)

returns names of actiove exogenizing dummies

sets the property self. exodummy_per which defines the time over which the dummies are defined

property fix_dummy_fixed

returns names of actiove exogenizing dummies

sets the property self. exodummy_per which defines the time over which the dummies are defined

property fix_dummy_fixed_old

returns names of actiove exogenizing dummies

sets the property self. exodummy_per which defines the time over which the dummies are defined

property fix_add_factor_fixed

Returns the add factors corrosponding to the active exogenizing dummies

property fix_value_fixed

Returns the exogenizing values corrosponding to the active exogenizing dummies

property fix_endo_fixed

Returns the endogeneous variables corrosponding to the active exogenizing dummies

fix_inf(*df=None*)

Display information regarding exogenizing

class modelclass.**model**(*i_eq=''*, *modelname='testmodel'*, *silent=False*, *straight=False*, *funks=[]*, *tabcomplete=True*, *previousbase=False*, *use_preorder=True*, *normalized=True*, *safeorder=False*, *var_description={}*, ***kwargs*)

This is the main model definition

class modelclass.**upd**(*pandas_obj*)

Extend a dataframe to update variables from string

look at `update` for syntax

__call__(*updates*, *lprint=False*, *scale=1.0*, *create=True*, *keep_growth=False*)

Call self as a function.

modelclass.**create_model**(*navn*, *hist=0*, *name=''*, *new=True*, *finished=False*, *xmodel=<class 'modelclass.model'>*, *straight=False*, *funks=[]*)

Creates either a model instance or a model and a historic model from formulars.

The formulars can be in a string or in af file withe the extension .txt

if:

Navn The model as text or as a file with extension .txt

Name Name of the model

New If True, ! used for comments, else () can also be used. False should be avoided, only for old PCIM models.

Hist If True, a model with calculations of historic value is also created

Xmodel The model class used for creating model the model instance. Can be used to create models with model subclasses

Finished If True, the model exploder is not used.

Straight If True, the formula sequence in the model will be used.

Funks A list of user defined funktions used in the model

`modelclass.`**`get_a_value`**(*df*, *per*, *var*, *lag=0*)

returns a value for row=p+lag, column = var

to take care of non additive row index

`modelclass.`**`set_a_value`**(*df*, *per*, *var*, *lag=0*, *value=nan*)

Sets a value for row=p+lag, column = var

to take care of non additive row index

`modelclass.`**`insertModelVar`**(*dataframe*, *model=None*)

Inserts all variables from model, not already in the dataframe. Model can be a list of models

`modelclass.`**`lineout`**(*vek*, *pre=''*, *w=20*, *d=0*, *pw=20*, *endline='\n'*)

Utility to return formated string of vector

`modelclass.`**`upddfold`**(*base*, *upd*)

takes two dataframes. The values from upd is inserted into base

`modelclass.`**`upddf`**(*base*, *upd*)

takes two dataframes. The values from upd is inserted into base

`modelclass.`**`randomdf`**(*df*, *row=False*, *col=False*, *same=False*, *ran=False*, *cpre='C'*, *rpre='R'*)

Randomize and rename the rows and columns of a dataframe, keep the values right:

Ran If True randomize, if False don't randomize

Col The columns are renamed and randdomized

Row The rows are renamed and randdomized

Same The row and column index are renamed and randdomized the same way

Cpre Column name prefix

Rpre Row name prefix

`modelclass.`**`join_name_lag`**(*df*)

creates a new dataframe where the name and lag from multiindex is joined as input a dataframe where name and lag are two levels in multiindex

`modelclass.`**`timer_old`**(*input='test'*, *show=True*, *short=False*)

does not catch exceptrions use model.timer

A timer context manager, implemented using a generator function. This one will report time even if an exception occurs"""

Parameters

- **input** (*string, optional*) – a name. The default is 'test'.
- **show** (*bool, optional*) – show the results. The default is True.
- **short** (*bool, optional*) – . The default is False.

Return type None.

4.2 Modelpattern, regular expressions to parse equations and models

Created on Mon Sep 02 19:32:22 2013

This module defines a number of pattern used in PYFS. If a new function is intruduced in the model definition language it should added to the function names in funkname

All functions in the module modeluserfunk will be added to the language and incorporated in the Business Logic language

@author: Ib

class modelpattern.**nterm**(*number*, *op*, *var*, *lag*)

lag

Alias for field number 3

number

Alias for field number 0

op

Alias for field number 1

var

Alias for field number 2

modelpattern.**find_frml**(*equations*)

Takes at modeltext and returns a list with where each element is a string starting with FRML and ending with $ It do not check if it is a valid FRML statement

modelpattern.**split_frml**(*frml*)

Splits a string with a frml into a tuple with 4 parts:

0. The unsplit frml statement
1. FRML
2. <Frml name>
3. <the frml expression>

modelpattern.**find_statements**(*a_model*)

splits a modeltest into comments and statements

- a *comment* starts with ! and ends at lineend
- a *statement* starts with a name and ends with a $ all characters between are considerd part of the statement

The statement is not chekked for meaningfulness returns a list of tuppels (comment,command,<rest of statement>)

`modelpattern.`**`model_parse_old`**`(`*equations*, *funks=[]*`)`

Takes a model returns a list of tupels. Each tupel contains:

the compleete formular

FRML

formular name

the expression

list of terms from the expression

The purpose of this function is to make model analysis faster. this is 20 times faster than looping over espressions in a model

`modelpattern.`**`model_parse`**`(`*equations*, *funks=[]*`)`

Takes a model returns a list of tupels. Each tupel contains:

the compleete formular

FRML

formular name

the expression

list of terms from the expression

The purpose of this function is to make model analysis faster. this is 20 times faster than looping over espressions in a model

This new model_parse handels lags of -0 or +0 which ocours in some models from world bank.

`modelpattern.`**`list_extract`**`(`*equations*, *silent=True*`)`

creates lists used in a model

returns a dictonary with the lists if a list is defined several times, the first definition is used

`modelpattern.`**`check_syntax_model`**`(`*equations*, *test=True*`)`

cheks if equations have syntax errors by calling the python compile.parse

`modelpattern.`**`udtryk_parse`**`(`*udtryk*, *funks=[]*`)`

returns a list of terms from an expression ie: lhs=rhs $ or just an expression like x+b

`modelpattern.`**`kw_frml_name`**`(`*frml_name0*, *kw*, *default=None*`)`

find keywords and associated value from string '<kw=xxx,res=kdkdk>'

4.3 Modelnewton, Newton solution and calculate derivatives

Created on Fri Jun 19 19:49:50 2020

@author: IBH

Module which handles model differentiation, construction of jacobi matrizex, and creates dense and sparse solving functions.

`class` `modelnewton.`**`diff_value_base`**`(`*var: str*, *pvar: str*, *lag: int*, *var_plac: int*, *pvar_plac: int*, *pvar_endo: bool*, *pvar_exo_plac: int*`)`

class define columns in database with values from differentiation class modelnewton.**diff_value_col**(*var: str*, *pvar: str*, *lag: int*, *var_plac: int*, *pvar_plac: int*, *pvar_endo: bool*, *pvar_exo_plac: int*)

The hash able class which can be used as pandas columns

class modelnewton.**diff_value**(*var: str*, *pvar: str*, *lag: int*, *var_plac: int*, *pvar_plac: int*, *pvar_endo: bool*, *pvar_exo_plac: int*, *number: int = 0*, *date: any = 0*)

class to contain values from differentiation

class modelnewton.**newton_diff**(*mmodel*, *df=None*, *endovar=None*, *onlyendocur=False*, *timeit=False*, *silent=True*, *forcenum=False*, *per=''*, *ljit=0*, *nchunk=None*, *endoandexo=False*)

Class to handle newron solving this is for un-nomalized or normalized models ie models of the forrm

0 = G(y,x) y = F(y,x)

modeldiff()

Differentiate relations for self.enovar with respect to endogeneous variable The result is placed in a dictory in the model instanse: model.diffendocur

show_diff(*pat='*'*)

Displays espressions for differential koifficients for a variable if var ends with * all matchning variables are displayes

show_stacked_diff(*time=None*, *lhs=''*, *rhs=''*, *dec=2*, *show=True*)

Parameters

- **time** (*list, optional*) – DESCRIPTION. The default is None. Time for which to retrieve stacked jacobi
- **lhs** (*string, optional*) – DESCRIPTION. The default is ‘’. Left hand side variables
- **rhs** (*TYPE, optional*) – DESCRIPTION. The default is ‘’. Right hand side variabnles
- **dec** (*TYPE, optional*) – DESCRIPTION. The default is 2.
- **show** (*TYPE, optional*) – DESCRIPTION. The default is True.

Return type selected rows and columns of stacked jacobi as dataframe .

get_diffmodel()

Returns a model which calculates the partial derivatives of a model

get_diff_melted(*periode=None*, *df=None*)

returns a tall matrix with all values to construct jacobimatrix(es)

get_diff_mat_tot(*df=None*)

Fetch a stacked jacobimatrix for the whole model.current_per

Returns a sparse matrix.

get_diff_mat_1per(*periode=None*, *df=None*)

fetch a dict of one periode sparse jacobimatrices

get_diff_melted_var(*periode=None*, *df=None*)

makes dict with all derivative matrices for all lags

get_diff_values_all(*periode=None*, *df=None*, *asdf=False*)

stuff the values of derivatives into nested dic

static **get_feedback**(*eig_dic*, *per=None*)

Returns a dict of max abs eigenvector and the sign

CHAPTER

FIVE

PROCESSING MODEL SPECIFICATION

The purpose is to process models specified in different ways - Macro business language - Eviews - Excel - Latex

and make them into the modelflows business Logic language.

5.1 Modelmanipulation, Text processing of of models before they become model class instances

Created on Mon Sep 02 19:41:11 2013

This module is a textprocessing module which is used to transforms a *template model* for a generic bank into into a unrolled and expande model which covers all banks - under control of a list feature.

The resulting model can be solved after beeing proccesed in the *modelclass* module.

In addition to creating a model for forecasting, the module can also create a model which calculates residulas and and variables in historic periods. This model can also be solved by the *modelclass* module.

@author: Ib

`class modelmanipulation.`**`safesub`**

A subclass of dict. if a *safesub* is indexed by a nonexisting keyword it just return the keyword this alows missing keywords when substitution text inspired by Python cookbook

`modelmanipulation.`**`sub`**(*text*, *katalog*)

Substitutes keywords from dictionary by returning text.format_map(safesub(katalog)) Allows missing keywords by using safesub subclass

`modelmanipulation.`**`oldsub_frml`**(*ibdic*, *text*, *plus=''*, *var=''*, *lig=''*, *sep='\n'*)

to repeat substitution from list

- *plus* is a seperator, used for creating sums
- *var* and *lig* Determins for which items the substitution should take place by var=abe, lig='ko' substitution is only performed for entries where var=='ko'

`modelmanipulation.`**`sub_frml`**(*ibdic*, *text*, *plus=''*, *xvar=''*, *lig=''*, *sep='\n'*)

to repeat substitution from list

- *plus* is a seperator, used for creating sums
- *xvar* and *lig* Determins for which items the substitution should take place by var=abe, lig='ko' substitution is only performed for entries where var=='ko'
- *xvar* is the variable to chek against selected in list

- *select list* is a list of elements in the xvar list to be included
- *matc* is the entry in *select list* from which to select from xvar

`modelmanipulation.`**`find_res`**`(`*f*`)`

Finds the expression which calculates the residual in a formel. FRML <res=a,endo=b> x=a*b+c $

`modelmanipulation.`**`find_res_dynare`**`(`*equations*`)`

equations to calculat _res formulas FRML <> x=a*b+c +x_RES $ -> FRML <> x_res =x-a*b+c $

`modelmanipulation.`**`find_res_dynare_new`**`(`*equations*`)`

equations to calculat _res formulas FRML <> x=a*b+c +x_RES $ -> FRML <> x_res =x-a*b+c $ not finished to speed time up

`modelmanipulation.`**`find_hist_model`**`(`*equations*`)`

takes a unrolled model and create a model which can be run for historic periode

and the identities are also calculeted

`modelmanipulation.`**`exounroll`**`(`*in_equations*`)`

takes a model and makes a new model by enhancing frml's with <exo=,j=,jr=> in their frml name.

Exo the value can be fixed in to a value valuename_x by setting valuename_d=1

Jled a additiv adjustment element is added to the frml

Jrled a multiplicativ adjustment element is added to the frml

`modelmanipulation.`**`tofrml`**`(`*expressions*, *sep='\n'*`)`

a function, wich adds FRML to all expressions seperated by <sep> if no start is specified the max lag will be used

`modelmanipulation.`**`dounloop`**`(`*in_equations*, *listin=False*`)`

Expands (unrolls do loops in a model template goes trough a model template until there is no more nested do loops

`modelmanipulation.`**`find_arg`**`(`*funk*, *streng*`)`

chops a string in 3 parts

1. before 'funk('
2. in the matching parantesis
3. after the last matching parenthesis

`modelmanipulation.`**`sumunroll_old`**`(`*in_equations*, *listin=False*`)`

expands all sum(list,'expression') in a model returns a new model

`modelmanipulation.`**`lagarray_unroll`**`(`*in_equations*, *funks=[]*`)`

expands all sum(list,'expression') in a model returns a new model

`modelmanipulation.`**`sumunroll`**`(`*in_equations*, *listin=False*`)`

expands all sum(list,'expression') in a model if sum(list xvar=lig,'expression') only list elements where the condition is satisfied wil be summed

returns a new model

`modelmanipulation.`**`argunroll`**`(`*in_equations*, *listin=False*`)`

expands all ARGEXPAND(list,'expression') in a model returns a new model `modelmanipulation.`**`creatematrix`**(*in_equations*, *listin=False*)

expands all ARGEXPAND(list,'expression') in a model returns a new model

`modelmanipulation.`**`createarray`**(*in_equations*, *listin=False*)

expands all to_array(list) in a model returns a new model

`modelmanipulation.`**`kaedeunroll`**(*in_equations*, *funks=[]*)

unrolls a chain (kaede) expression - used in the SMEC moedel

`modelmanipulation.`**`check_syntax_frml`**(*frml*)

check syntax of frml

`modelmanipulation.`**`normalize_a_frml`**(*frml*, *show=False*)

Normalize and show a frml

`modelmanipulation.`**`nomalize_a_model`**(*equations*)

a symbolic normalization is performed if there is a syntaxerror

`modelmanipulation.`**`normalize`**(*in_equations*, *sym=False*, *funks=[]*)

Normalize an equation with log or several variables at the left hand side, the first variable is considerd the endogeneeus

`modelmanipulation.`**`explode`**(*model*, *norm=True*, *sym=False*, *funks=[]*, *sep='\n'*)

prepares a model from a model template.

Returns a expanded model which is ready to solve

Eksempel: model = udrul_model(MinModel.txt)

`modelmanipulation.`**`modelprint`**(*ind*, *title=' A model'*, *udfil=''*, *short=0*)

prettyprinter for a a model. :udfil: if present is output file :short: if present condences the model Can handle both model templates and models

`modelmanipulation.`**`lagone`**(*ind*, *funks=[]*, *laglead=- 1*)

All variables in a string i s lagged one more time

`modelmanipulation.`**`lag_n_tup`**(*udtryk*, *n=- 1*, *funks=[]*)

return a tuppel og lagged expressions from lag = 0 to lag = n)

`modelmanipulation.`**`pastestring`**(*ind*, *post*, *funks=[]*, *onlylags=False*)

All variable names in a in a string **ind** is pasted with the string **post**

This function can be used to awoid variable name conflict with the internal variable names in sympy.

an advanced function

`modelmanipulation.`**`stripstring`**(*ind*, *post*, *funks=[]*)

All variable names in a in a string is **ind** is stripped of the string **post**.

This function reverses the pastestring process

`modelmanipulation.`**`findindex`**(*ind00*)

find the index variables meaning variables on the left hand side of = braced by {}

`modelmanipulation.`**`doablelist`**(*expressions*, *sep='\n'*)

create a list of tupels from expressions seperated by sep, each element in the list is a tupel (index, number og expression, the expression)

we want make group the expressions acording to index index is elements on the left of = braced by {}

`modelmanipulation.`**`doablekeep`**`(`*formulars*`)`

takes index in the lhs and creates a do loop around the lines with same indexes on the right side you can use %0_, %1_ an so on to indicate the index, just to awoid typing to much

Also %i_ will be changed to all the indexes

`modelmanipulation.`**`doable`**`(`*formulars*, *funks=[]*`)`

takes index in the lhs and creates a do loop around the line on the right side you can use %0_, %1_ an so on to indicate the index, just to awoid typing to much

Also %i_ will be changed to all the indexes

`modelmanipulation.`**`findindex_gams`**`(`*ind00*`)`

- an equation looks like this
- <frmlname> [index] lhs = rhs

this function find frmlname and index variables on the left hand side. meaning variables braced by {}

`modelmanipulation.`**`un_normalize_expression`**`(`*frml*`)`

This function makes sure that all formulas are unnormalized. if the formula is already decorated with <endo=name> this is kept else the lhs_varriable is used in <endo=>

`modelmanipulation.`**`un_normalize_model`**`(`*in_equations*, *funks=[]*`)`

un normalize a model

`modelmanipulation.`**`un_normalize_simpel`**`(`*in_equations*, *funks=[]*`)`

un-normalize expressions delimeted by linebreaks

`modelmanipulation.`**`eksempel`**`(`*ind*`)`

takes a template model as input, creates a model and a histmodel and prints the models

5.2 Modelnormalize, Transforms and normalizes single equations

Created on Sat Nov 28 13:32:47 2020

This Module is used transforming model specifications to modelflow business language.

- **preprocessing expressions to resolve functions like** dlog, log, pct, movavg
- replace function names
- normalize formulas

@author: bruger

`class modelnormalize.`**`Normalized_frml`**`(`*endo_var: str = '', original: str = '', preprocessed: str = '', normalized: str = '', calc_add_factor: str = '', un_normalized: str = '', fitted: str = '', eviews: str = ''*`)`

class defining result from normalization of expression

`modelnormalize.`**`endovar`**`(`*f*`)`

Finds the first variable in a expression

`modelnormalize.`**`funk_in`**`(`*funk*, *a_string*`)`

Find the first location of a function in a string

if found returns a match object where the group 2 is the interesting stuff used in funk_find_arg `modelnormalize.`**`funk_replace`**`(`*funk1*, *funk2*, *a_string*`)`

replace funk1(with funk2(

takes care that funk1 embedded in variable name is not replaced

`modelnormalize.`**`funk_replace_list`**`(`*replacelist*, *a_string*`)`

Replaces a list of funk1(, funk2(

`modelnormalize.`**`funk_find_arg`**`(`*funk_match*, *streng*`)`

chops a string in 3 parts

1. before 'funk('
2. in the matching parantesis
3. after the last matching parenthesis

`modelnormalize.`**`preprocess`**`(`*udtryk*, *funks=[]*`)`

test processing expanding dlog,diff,movavg,pct,logit functions

Parameters

- **`udtryk`** (*`str`*) – model we want to do template expansion on
- **`funks`** (*`list, optional`*) – list of user defined functions . Defaults to [].

Returns None.

has to be changed to (= for when the transition to 3.8 is finished.

`modelnormalize.`**`normal`**`(`*ind_o*, *the_endo=''*, *add_add_factor=True*, *do_preprocess=True*, *add_suffix='_A'*, *endo_lhs=True*, *make_fixable=False*, *make_fitted=False*, *eviews=''*`)`

normalize an expression g(y,x) = f(y,x) ==> y = F(x,z)

Default find the expression for the first variable on the left hand side (lhs)

The variable - without lags- should not be on rhs.

Parameters

- **`ind_o`** (*`str`*) – input expression, no $ and no frml name just lhs=rhs
- **`the_endo`** (*`str, optional`*) – the endogeneous to isolate on the left hans side. if the first variable in the lhs. It shoud be on the left hand side.
- **`add_add_factor`** (*`bool, optional`*) – force introduction aof adjustment term, and an expression to calculate it
- **`do_preprocess`** (*`bool, optional`*) – DESCRIPTION. preprocess the expression
- **`endo_lhs`** (*`bool, optional`*) – If false, accept to normalize for a rhs endogeneous variable
- **`make_fixable`** (*`bool, optional`*) – also make this equation exogenizable
- **`fitted`** (*`bool,optional`*) – create a fitted equations, without exo and adjustment

preprocessing handels

Returns Normalized_frml which will contain the different relevant expressions

Return type An instance of the class

`modelnormalize.`**`elem_trans`**`(`*udtryk*, *df=None*`)`

Handeles expression with @elem

5.3 Onboarding models

Modules to onboard models from different sources.

The process of onboarding involves transforming the original specification to **Modelflow Business Logic Language** using what ever tools needed. As Python has very powerfull string and datatools it is possible to onboard many models - but by all means not all models.

Be aware, that the functions presented here are made for specific model(families) following specific conventions. If these conventions are not followed, another model can't be onboarded

5.3.1 Eviews

5.3.1.1 From wf1 file

Created on Wed Mar 30 10:06:26 2022

@author: ibhan

Module to handle models in wf1 files

1. Eviews is started and the wf1 file is loaded.
    1. Some transformations are performed on data.
    2. The model is unlinked.
    3. The workspace is saved as a wf2 file. Same name with _modelflow appended.
2. Eviews is closed
3. The wf2 file is read as a json file.
4. Relevant objects are extracted.
5. The MFMSA variable is extracted, to be saved in the dumpfile.
6. The equations are transformed and normalized to modelflow format and classified into identities and stochastic
7. Stochastic equations are enriched by add_factor and fixing terms (dummy + fixing value)
8. For Stochastic equations new fitted variables are generated - without add add_factors and dummies.
9. A model to generate fitted variables is created
10. A model to generate add_factors is created.
11. A model encompassing the original equations, the model for fitted variables and for add_factors is created.
12. The data series and scalars are shoveled into a Pandas dataframe
    1. Some special series are generated as the expression can not be incorporated into modelflow model specifications
    2. The model for fitted values is simulated in the specified timespan
    3. The model for add_factors is simulated in the timespan set in MFMSA
13. The data descriptions are extracted into a dictionary.
14. Data descriptions for dummies, fixed values, fitted values and add_factors are derived.
15. Now we have a model and a dataframe with all variables which are needed.

`modelgrabwf2.`**`wf1_to_wf2`**(*filename*, *modelname=''*, *eviews_run_lines=[]*)

- Opens a eviews workfile in wf1 format
- calculates the eviews_trend
- unlink a model and
- writes the workspace back to a wf2 file

Parameters

- **`filename`** (*`TYPE`*) – DESCRIPTION.
- **`modelname`** (*`TYPE`*) – default ‘’ then the three first letters of the filenames stem are asumed to be the modelname.

Returns None.

`modelgrabwf2.`**`wf2_to_clean`**(*wf2name*, *modelname=''*, *save_file=False*)

Takes a eviews .wf2 file - which is in JSON format - and place a dictionary

Parameters

- **`wf2name`** (*`TYPE`*) – name of wf2 file .
- **`modelname`** (*`TYPE, optional`*) – Name og model. Defaults to ‘’.
- **`save_file`** (*`TYPE, optional`*) – save the specification, data and description in a dictionary. Defaults to False.

Returns the content of the wf2 file as a dict .

Return type model_all_about (dict)

`class modelgrabwf2.`**`GrabWfModel`**(*filename: any = ''*, *modelname: any = ''*, *eviews_run_lines: list = <factory>*, *model_all_about: dict = <factory>*, *start: typing.Optional[any] = None*, *end: typing.Optional[any] = None*, *country_trans: any = <function GrabWfModel.<lambda>>*, *country_df_trans: any = <function GrabWfModel.<lambda>>*, *make_fitted: bool = False*, *fit_start: any = 2000*, *fit_end: typing.Optional[any] = None*, *do_add_factor_calc: bool = True*, *test_frml: str = ''*, *disable_progress: bool = False*)

This class takes a world bank model specification, variable data and variable description and transform it to ModelFlow business language

Parameters

- **`filename`** – any = ‘’ #wf1 name
- **`modelname`** – any = ‘’
- **`eviews_run_lines`** – list =field(default_factory=list)
- **`model_all_about`** – dict = field(default_factory=dict)
- **`start`** – any = None # start of testing if not overruled by mfmsa
- **`end`** – any = None # end of testing if not overruled by mfmsa
- **`country_trans`** – any = lambda x:x[:] # function which transform model specification
- **`country_df_trans`** – any = lambda x:x # function which transforms initial dataframe
- **`make_fitted`** – bool = False # if True, a clean equation for fittet variables is created
- **`fit_start`** – any = 2000 # start of fittet model

- `fit_end` – any = None # end of fittet model unless overruled by mfmsa
- `do_add_factor_calc` – bool = True # calculate the add factors
- `test_frml` – str =” # a testmodel as string if used no wf processing
- `disable_progress` – bool = False # Disable progress bar

`__post_init__()`

Process the model

`static trans_eviews(`*rawmodel*`)`

Takes Eviews specifications and wrangle them into modelflow specifications

Parameters `rawmodel` (*TYPE*) – a raw model .

Returns a model with the appropiate eviews transformations.

Return type rawmodel6 (TYPE)

`property var_description`

Adds var descriptions for add factors, exogenizing dummies and exoggenizing values

`property mfmsa_options`

Grab the mfmsa options, a world bank speciality

`property mfmsa_start_end`

Finds the start and end from the MFMSA entry

`property dfmodel`

The original input data enriched with during variablees, variables containing values for specific historic years and model specific transformation

`test_model(`*start=None*, *end=None*, *maxvar=1000000*, *maxerr=100*, *tol=0.0001*, *showall=False*, *showinput=False*`)`

Compares a straight calculation with the input dataframe.

shows which variables dont have the same value

Parameters

- `df` (*TYPE*) – dataframe to run.
- `start` (*TYPE, optional*) – start period. Defaults to None.
- `end` (*TYPE, optional*) – end period. Defaults to None.
- `maxvar` (*TYPE, optional*) – how many variables are to be chekked. Defaults to 1_000_000.
- `maxerr` (*TYPE, optional*) – how many errors to check Defaults to 100.
- `tol` (*TYPE, optional*) – check for absolute value of difference. Defaults to 0.0001.
- `showall` (*TYPE, optional*) – show more . Defaults to False.
- `showinput` (*TYPE, optional*) – show the input values Defaults to False.

Returns None.

CHAPTER

SIX

PROCESSING RESULTS

6.1 Modelvis, Display and vizualize variables

Created on Fri May 12 11:07:02 2017

@author: hanseni

This module creates functions and classes for visualizing results.

`modelvis.`**`meltdim`**`(`*df*, *dims=['dima', 'dimb']*, *source='Latest'*`)`

Melts a wide dataframe the variable names are split to dimensions acording to the list of texts in dims. in variablenames the tall dataframe have a variable name for each dimensions also values and source are introduced ac column names in the dataframe

`class` `modelvis.`**`vis`**`(`*model=None*, *pat=''*, *names=None*, *df=None*`)`

Visualization class. used as a method on a model instance.

The purpose is to select variables according to a pattern, potential with wildcards

`heat``(`**args*, ***kwargs*`)`

Displays a heatmap of the resulting dataframe

`plot``(`**args*, ***kwargs*`)`

Displays a plot for each of the columns in the resulting dataframe

`plot_alt``(`*title='Title'*, **args*, ***kwargs*`)`

Displays a plot for each of the columns in the resulting dataframe

`box``()`

Displays a boxplot comparing basedf and lastdf

`violin``()`

Displays a violinplot comparing basedf and lastdf

`swarm``()`

Displays a swarmlot comparing basedf and lastdf

`property df`

Returns the result of this instance as a dataframe

`property base`

Returns basedf

`property pct`

Returns the pct change

property year_pct

Returns the pct change over 4 periods (used for quarterly data)

property frml

Returns formulas

property des

Returns variable descriptions

property dif

Returns the differens between the basedf and lastdf

property difpctlevel

Returns the differens between the basedf and lastdf

property difpct

Returns the differens between the pct changes in basedf and lastdf

property print

prints the current result

__repr__()

Return repr(self).

_repr_html_()

Displays a nice summary of the results when called in a Jupyter envioreement

rename(*other=None*)

rename columns

mul(*other*)

Multiply the curent result with other

property mul100

Multiply the current result with 100

class modelvis.**compvis**(*model=None*, *pat=None*)

Class to compare to runs in boxplots

box(**args*, ***kwargs*)

Displays a boxplot

swarm(**args*, ***kwargs*)

Displays a swarmplot

violin(**args*, ***kwargs*)

Displays a violinplot

class modelvis.**container**(*lastdf*, *basedf*)

A container, used if to izualize dataframes without a model

smpl(*start=''*, *slut=''*, *df=None*)

Defines the model.current_per which is used for calculation period/index when no parameters are issues the current current period is returned

Either none or all parameters have to be provided

vlist(*pat*)

returns a list of variable matching the pattern class modelvis.**varvis**(*model=None*, *var=''*)

Visualization class. used as a method on a model instance.

The purpose is to select variables acording to a pattern, potential with wildcards

tracedep(*down=1*, ***kwargs*)

Trace dependensies of name down to level down

tracepre(*up=1*, ***kwargs*)

Trace dependensies of name down to level down

__repr__()

Return repr(self).

modelvis.**vis_alt**(*grund*, *mul*, *title='Show variables'*, *top=0.9*)

Graph of one of more variables each variable is displayed for 3 banks

modelvis.**plotshow**(*df*, *name=''*, *ppos=- 1*, *kind='line'*, *colrow=2*, *sharey=False*, *top=0.9*, *splitchar='__'*, *savefig=''*, **args*, ***kwargs*)

Parameters

- **df** (*TYPE*) – Dataframe .
- **name** (*TYPE, optional*) – title. Defaults to ‘’.
- **ppos** (*TYPE, optional*) – # of position to use if split. Defaults to -1.
- **kind** (*TYPE, optional*) – matplotlib kind . Defaults to ‘line’.
- **colrow** (*TYPE, optional*) – columns per row . Defaults to 6.
- **sharey** (*TYPE, optional*) – Share y axis between plots. Defaults to True.
- **top** (*TYPE, optional*) – relative position of the title. Defaults to 0.90.
- **splitchar** (*TYPE, optional*) – if the name should be split . Defaults to ‘__’.
- **savefig** (*TYPE, optional*) – save figure. Defaults to ‘’.
- **xsize** (*TYPE, optional*) – x size default to 10
- **ysize** (*TYPE, optional*) – y size per row, defaults to 2

Returns a matplotlib fig.

modelvis.**melt**(*df*, *source='Latest'*)

melts a wide dataframe to a tall dataframe , appends a soruce column

modelvis.**heatshow**(*df*, *name=''*, *cmap='Reds'*, *mul=1.0*, *annot=False*, *size=(11.69, 8.27)*, *dec=0*, *cbar=True*, *linewidths=0.5*)

A heatmap of a dataframe

modelvis.**attshow**(*df*, *treshold=False*, *head=5000*, *tail=0*, *t=True*, *annot=False*, *showsum=False*, *sort=True*, *size=(11.69, 8.27)*, *title=''*, *tshow=True*, *dec=0*, *cbar=True*, *cmap='jet'*, *savefig=''*)

Shows heatmap of impacts of exogeneous variables :df: Dataframe with impact :treshold: Take exogeneous variables with max impact of treshold or larger :numhigh: take the numhigh largest impacts :t: transpose the heatmap :annot: Annotate the heatmap :head: take the head largest .tail: take the tail smalest :showsum: Add a column with the sum :sort: Sort the data .tshow: Show a longer title :cbar: if a colorbar shoud be displayes :cmap: the colormap :save: Save the chart (in png format) `modelvis.`**`attshowone`**(*df*, *name*, *pre=''*, *head=5*, *tail=5*)

shows the contribution to row=name from each column the coulumns can optional be selected as starting with pre

`modelvis.`**`water`**(*serxinput*, *sort=False*, *ascending=True*, *autosum=False*, *allsort=False*, *threshold=0.0*)

Creates a dataframe with information for a watrfall diagram

Serx the input serie of values

Sort True if the bars except the first and last should be sorted (default = False)

Allsort True if all bars should be sorted (default = False)

Autosum True if a Total bar are added in the end

Ascending True if sortorder = ascending

Returns a dataframe with theese columns:

Hbegin Height of the first bar

Hend Height of the last bar

Hpos Height of positive bars

Hneg Height of negative bars

Start Ofset at which each bar starts

Height Height of each bar (just for information)

CHAPTER SEVEN

ATTRIBUTION

7.1 Equation level

Attribution can be performed on the equation level and on the model level

Equation level attribution is done in the modelclass module here `Dekomp_Mixin`

The class `Dekomp_Mixin` also defines a a number of front end functions both for equation and model attribution

7.2 Model level

Module for making attribution analysis of a model.

The main function is attribution

Created on Wed May 31 08:50:51 2017

@author: hanseni

`modeldekom.`**`attribution`**(*model*, *experiments*, *start=''*, *end=''*, *save=''*, *maxexp=10000*, *showtime=False*, *summaryvar=['*']*, *silent=False*, *msilent=True*, *type='level'*)

> Calculates an attribution analysis on a model accepts a dictionary with experiments. the key is experiment name, the value is a list of variables which has to be reset to the values in the baseline dataframe.

`modeldekom.`**`attribution_new`**(*model*, *experiments*, *start=''*, *end=''*, *save=''*, *maxexp=10000*, *showtime=False*, *summaryvar=['*']*, *silent=False*, *msilent=True*, *type='level'*)

> Calculates an attribution analysis on a model accepts a dictionary with experiments. the key is experiment name, the value is a list of variables which has to be reset to the values in the baseline dataframe.

`modeldekom.`**`ilist`**(*df*, *pat*)

> returns a list of variable in the model matching the pattern, the pattern can be a list of patterns of a sting with patterns seperated by blanks
>
> This function operates on the index names of a dataframe. Relevant for attribution analysis

`modeldekom.`**`GetSumImpact`**(*impact*, *pat='PD__*'*)

> Gets the accumulated differences attributet to each impact group

`modeldekom.`**`GetLastImpact`**(*impact*, *pat='RCET1__*'*)

> Gets the last differences attributet to each impact group

`modeldekom.`**`GetAllImpact`**(*impact*, *pat='RCET1__*'*)

> Gets the last differences attributet to each impact group

modeldekom.**GetOneImpact**(*impact*, *pat='RCET1__*'*, *per=''*)

Gets differences attributet to each impact group in period:per

modeldekom.**AggImpact**(*impact*)

Calculates the sum of impacts and place in the last column

This function is applied to the result iof a Get* function

class modeldekom.**totdif**(*model*, *summaryvar='*'*, *desdic={}*, *experiments=None*)

Class to make modelvide attribution analysis

explain_last(*pat=''*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*)

Explains last period

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘level’.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.

Returns DESCRIPTION.

Return type fig (TYPE)

explain_sum(*pat=''*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*)

Explains the sum

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘level’.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.

Returns DESCRIPTION.

Return type fig (TYPE)

explain_per(*pat=''*, *per=''*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*, *ysize=5*)

Explains a periode

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **per** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘level’.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.
- **ysize** (*TYPE, optional*) – DESCRIPTION. Defaults to 5.

Returns DESCRIPTION.

Return type fig (TYPE)

`explain_all`(*pat=''*, *stacked=True*, *kind='bar'*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*, *resample=''*, *axvline=None*)

Explains all

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **stacked** (*TYPE, optional*) – DESCRIPTION. Defaults to True.
- **kind** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘bar’.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘level’.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.
- **resample** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **axvline** (*TYPE, optional*) – DESCRIPTION. Defaults to None.

Returns None.

`totexplain`(*pat='*'*, *vtype='all'*, *stacked=True*, *kind='bar'*, *per=''*, *top=0.9*, *title=''*, *use='level'*, *threshold=0.0*, *ysize=10*, ***kwargs*)

Wrapper for different explanations

- *explain_last*
- *explain_per*
- *explain_sum*
- *explain_all*

Parameters

- **pat** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘*’.
- **vtype** (*per|all|last|sum, optional*) – what data to attribute. Defaults to ‘all’.
- **stacked** (*TYPE, optional*) – DESCRIPTION. Defaults to True.
- **kind** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘bar’.
- **per** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **top** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.9.
- **title** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **use** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘level’.
- **threshold** (*TYPE, optional*) – DESCRIPTION. Defaults to 0.0.
- **ysize** (*TYPE, optional*) – DESCRIPTION. Defaults to 10.
- ****kwargs** (*TYPE*) – DESCRIPTION.

Returns DESCRIPTION.

Return type fig (TYPE)

CHAPTER

EIGHT

TARGETS AND INSTRUMENTS

Used from the model class here *invert*

Created on Thu Sep 21 12:41:10 2017

@author: IBH

Class to handle general target/instrument problems.

Number of targets should be equal to number of instruments

An instrument can comprice of severeral variables instruments are inputtet as a list of instruments

class modelinvert.**targets_instruments**(*databank*, *targets*, *instruments*, *model*, *DefaultImpuls=0.01*, *defaultconv=0.01*, *nonlin=False*, *silent=True*, *maxiter=30*, *solveopt={}*, *varimpulse=False*)

Class to handle general target/instrument problems. Where the response is delayed specify this with delay.

Number of targets should be equal to number of instruments

An instrument can comprice of severeral variables

Instruments are inputtet as a list of instruments

To calculate the jacobian each instrument variable has a impuls, which is used as delta when evaluating the jacobi matrix:

```
[ 'QO_J','TG']   Simple list each variable are shocked by the default impulse
[ ('QO_J',0.5), 'TG']  Here QO_J is getting its own impuls (0.5)
[ [('QO_J',0.5),('ORLOV',1.)] , ('TG',0.01)] here an impuls is given for each␣
↪variable, and the first instrument consiste of two variables
```

Targets are list of variables

Convergence is achieved when all targets are within convergens distance from the target value

Convergencedistance can be set individual for a target variable by setting a value in <modelinstance>.targetconv

Targets and target values are provided by a dataframe.

jacobi(*per*, *delay=None*)

Calculates a jecobi matrix of derivatives based on the instruments and targets

returns a dataframe

invjacobi(*per*, *diag=False*, *delay=0*)

Calculates the inverted jacobi matrix

returns a dataframe

`targetseek`(*databank=None*, *shortfall=False*, *ti_damp=1.0*, *delay=0*, *progressbar=True*, ***kwargs*)

Calculates the instruments as a function of targets

CHAPTER

NINE

ENRICHING PANDAS DATAFRAMES

Pandas dataframes can be enriched and this is done in two instances. - to make it convinient to update variables - to embed modelflow into dataframes

9.1 When modelclass is imported upd is embedded

dataframes are equiped with the upd method. This allows convinient updating of variables using the `update` method.

9.2 When modelmf is imported, modelflow is imbedded dataframes

This is a module for extending pandas dataframes with the modelflow toolbox

Created on Sat March 2019

@author: hanseni

class `modelmf.`**`mf`**(*pandas_obj*)

A class to extend Pandas Dataframes with ModelFlow functionalities

Not to be used on its own

`copy`()

copy a modelflow extended dataframe, so it remember its model and options

`solve`(*start=''*, *slut=''*, ***kwargs*)

Solves a model

`makemodel`(*eq*, ***kwargs*)

Makes a model from equations

`__call__`(*eq=''*, ***kwargs*)

returns a model instace

Parameters

- **eq** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- ****kwargs** (*TYPE*) – DESCRIPTION.

Returns a instance .

Return type *model*

`__getitem__`(*name*)

Retrieves __getitem__ from model instance

class modelmf.**mfcalc**(*pandas_obj*)

Used to carry out calculation specified as equations

Parameters

- **eq** (*TYPE*) – Equations one on each line. can be started with <start end> to control calculation sample .
- **start** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **slut** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **showeq** (*TYPE, optional*) – If True the equations will be printed. Defaults to False.
- ****kwargs** (*TYPE*) – Here all solve options can be provided.

Returns Dataframe.

`__call__`(*eq*, *start=''*, *slut=''*, *showeq=False*, ***kwargs*)

This call performs the calculation

Parameters

- **eq** (*TYPE*) – Equations one on each line. can be started with <start end> to control calculation sample .
- **start** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **slut** (*TYPE, optional*) – DESCRIPTION. Defaults to ‘’.
- **showeq** (*TYPE, optional*) – If True the equations will be printed. Defaults to False.
- ****kwargs** (*TYPE*) – Here all solve options can be provided.

Returns Dataframe.

class modelmf.**mfupdate**(*pandas_obj*)

Extend a dataframe to update with values from another dataframe

`__call__`(*df*)

Call self as a function.

class modelmf.**ibloc**(*pandas_obj*)

Extend a dataframe with a slice method which accepot wildcards in column selection.

The method just juse the method vlist from modelclass.model class

`__call__`()

Not in use

CHAPTER

TEN

LOGICAL STRUCTURE

10.1 modelnet

Created on Wed Oct 15 14:30:44 2014

Displays an adjencacy matrix . @author: ibh

`modelnet.`**`draw_adjacency_matrix`**`(`*G*, *node_order=None*, *partitions=None*, *type=False*, *title='Structure'*, *size=(10, 10)*`)`

- G is a netorkx graph
- **node_order (optional) is a list of nodes, where each node in G** appears exactly once
- **partitions is a list of node lists, where each node in G appears** in exactly one node list
- type is a list of saying "simultaneous" or something else, has to have same length as partitions

`modelnet.`**`drawendoexo`**`(`*model*, *size=(6.0, 6.0)*`)`

Draw dependency including exogeneous. Used for illustrating for small models

CHAPTER

ELEVEN

JUPYTER STUFF

11.1 update widgets

Created on Mon Aug 9 14:46:11 2021

To define Jupyter widgets to update and show variables. @author: Ib

class modelwidget.**basewidget**(*datachildren: list = <factory>*)

basis for widget updating in jupyter

update_df(*df*, *current_per*)

will update container widgets

class modelwidget.**tabwidget**(*tabdefdict: dict*, *tab: bool = True*, *selected_index: Optional[any] = None*)

A widget to create tab or acordium contaners

update_df(*df*, *current_per*)

will update container widgets

reset(*g*)

will reset container widgets

class modelwidget.**sheetwidget**(*df_var: any = Empty DataFrame Columns: [] Index: []*, *trans: any = <function sheetwidget.<lambda>>*, *transpose: bool = False*, *expname: str = 'Carbon tax rate*, *US$ per tonn '*)

class defining a widget which updates from a sheet

class modelwidget.**slidewidget**(*slidedef: dict*, *altname: str = 'Alternative'*, *basename: str = 'Baseline'*, *expname: str = 'Carbon tax rate, US$ per tonn '*)

class defefining a widget with lines of slides

update_df(*df*, *current_per*)

updates a dataframe with the values from the widget

set_slide_value(*g*)

updates the new values to the self.current_vlues will be used in update_df

class modelwidget.**sumslidewidget**(*slidedef: dict*, *maxsum: Optional[any] = None*, *altname: str = 'Alternative'*, *basename: str = 'Baseline'*, *expname: str = 'Carbon tax rate, US$ per tonn '*)

class defefining a widget with lines of slides

update_df(*df*, *current_per*)

updates a dataframe with the values from the widget

set_slide_value(*g*)

updates the new values to the self.current_vlues will be used in update_df

class modelwidget.**updatewidget**(*mmodel: any*, *a_datawidget: any*, *basename: str = 'Business as usual'*, *keeppat: str = '*'*, *varpat: str = '*'*, *showvarpat: bool = True*, *exodif: any = Empty DataFrame Columns: [] Index: []*, *lwrun: bool = True*, *lwupdate: bool = False*, *lwreset: bool = True*, *lwsetbas: bool = True*, *lwshow: bool = True*, *outputwidget: str = 'jupviz'*, *prefix_dict: dict = <factory>*, *display_first: typing.Optional[any] = None*, *vline: list = <factory>*, *relativ_start: int = 0*, *short: bool = False*, *legend: bool = False*)

class to input and run a model

class modelwidget.**htmlwidget_df**(*mmodel: any*, *df_var: any = Empty DataFrame Columns: [] Index: []*, *trans: any = <function htmlwidget_df.<lambda>>*, *transpose: bool = False*, *expname: str = ''*, *percent: bool = False*)

class displays a dataframe in a html widget

class modelwidget.**htmlwidget_fig**(*figs: any*, *expname: str = ''*, *format: str = 'svg'*)

class displays a dataframe in a html widget

class modelwidget.**htmlwidget_label**(*expname: str = ''*, *format: str = 'svg'*)

class displays a dataframe in a html widget

class modelwidget.**visshow**(*mmodel: <built-in function any>*, *varpat: str = '*'*, *showvarpat: bool = True*, *show_on: bool = True*)

11.2 modeljupytermagic, Defines magic functions to define models, data and graphs in jupyter cells

This module defines several magic jupyter functions:

graphviz Draw Graphviz graph

dataframe Create Pandas Dataframe

latexflow Create a modelflow modelinstance from latex script

To display the doc strings use the functions in jupyter.

modeljupytermagic.**get_options**(*line*, *defaultname='test'*)

Retrives options from the first line

Parameters

- **line** (*TYPE*) – DESCRIPTION.
- **defaultname** (*TYPE, optional*) – DESCRIPTION. Defaults to 'test'.

Returns name . opt (dict): options.

Return type name (string)

CHAPTER

TWELVE

OPTIMIZATON

12.1 model_cvx

Created on Mon May 26 21:11:18 2014

@author: Ib Hansen

A good explanation of quadradic programming in cvxopt is in http://courses.csail.mit.edu/6.867/wiki/images/a/a7/Qp-cvxopt.pdf

This exampel calculates the efficient forntier in a small example the example is based on a mean variance model for Indonesian Rupia running in Excel

`model_cvx.mv_opt(`*PP*, *qq*, *riskaversion*, *bsum*, *weights*, *weigthtedsum*, *boundsmin*, *boundsmax*, *lprint=False*, *solget=None*`)`

Performs mean variance optimization by calling a quadratic optimization function from the cvxopt

library

`model_cvx.mv_opt_bs(`*msigma*, *vreturn*, *riskaversion*, *budget*, *risk_weights*, *capital*, *lcr_weights*, *lcr*, *leverage_weights*, *equity*, *boundsmin*, *boundsmax*, *lprint=False*, *solget=None*`)`

performs balance sheet optimization

`model_cvx.mv_opt_prop(`*PP*, *qq*, *riskaversion*, *bsum*, *weights*, *weigthtedsum*, *boundsmin*, *boundsmax*, *probability=None*, *lprint=False*`)`

select a numner of assets/liabilities which. when the selection is feasible an Mean variance optimazation is performed

the selection is based on probabilities

PYTHON MODULE INDEX

INDEX