

# **IRIS Toolbox Reference Manual**

Version 20120425

Runs in Matlab® R2010a or later

by

*Jaromír Beneš*

25 April 2012



## Preface

IRIS is a Matlab<sup>®</sup> toolbox for macroeconomic modelling. It is free, open-source software distributed under BSD licence terms. The current version of IRIS runs in Matlab R2009a or later.

IRIS has been designed as an integrated and extensible package. It includes not only objects and functions directly related to modelling itself, but also provides support for building and implementing model-aided forecasting and policy analysis systems, and for model-based economic research. This includes multivariate time series analysis (VARs, BVARs, FAVARs), basic time series and database management, and reporting.

# Contents

<b>Part I —</b>	<b>IRIS sessions</b>	<b>7</b>
<b>1</b>	<b>Installing IRIS</b>	<b>8</b>
<b>2</b>	<b>Starting, quitting, and configuring IRIS</b>	<b>10</b>
<b>3</b>	<b>Getting on-line help</b>	<b>20</b>
<b>Part II —</b>	<b>Modelling</b>	<b>22</b>
<b>4</b>	<b>Model file language</b>	<b>23</b>
<b>5</b>	<b>Model objects and functions</b>	<b>63</b>
<b>6</b>	<b>Simulation plans</b>	<b>137</b>
<b>7</b>	<b>Posterior objects and functions</b>	<b>148</b>
<b>8</b>	<b>Probability distribution package</b>	<b>155</b>
<b>9</b>	<b>Steady-state file language</b>	<b>161</b>
<b>10</b>	<b>Steady-state objects and functions</b>	<b>166</b>
<b>11</b>	<b>Matrices with named rows and columns</b>	<b>169</b>
<b>Part III —</b>	<b>Multivariate time series analysis</b>	<b>172</b>
<b>12</b>	<b>Vector autoregressions: VAR objects and functions</b>	<b>173</b>
<b>13</b>	<b>Structural vector autoregressions: SVAR objects and functions</b>	<b>201</b>
<b>14</b>	<b>Bayesian VAR prior dummies: BVAR package</b>	<b>211</b>

<b>15 Factor-augmented vector autoregressions: FAVAR objects and functions</b>	<b>214</b>
 <b>Part IV — Time series and database management</b>	 <b>224</b>
<b>16 Dates and date ranges</b>	<b>225</b>
<b>17 Time series objects and functions</b>	<b>247</b>
<b>18 Basic database management</b>	<b>311</b>
 <b>Part V — Reporting and publishing</b>	 <b>334</b>
<b>19 Report objects and functions</b>	<b>335</b>
<b>20 Quick-report file language</b>	<b>366</b>
<b>21 Quick-report functions</b>	<b>372</b>
<b>22 Graphics functions</b>	<b>377</b>

## Preface

IRIS is a Matlab® toolbox for macroeconomic modelling. It is free, open-source software distributed under BSD licence terms. The current version of IRIS runs in Matlab R2009a or later.

IRIS has been designed as an integrated and extensible package. It includes not only objects and functions directly related to modelling itself, but also provides support for building and implementing model-aided forecasting and policy analysis systems, and for model-based economic research. This includes multivariate time series analysis (VARs, BVARs, FAVARs), basic time series and database management, and reporting.

Part I —  
**IRIS sessions**

## 1 Installing IRIS

### Requirements

- Matlab R2009a or later.

### Optional components

*Optimization Toolbox* The Optimization Toolbox is needed to compute the steady state of non-linear models, and to run estimation.

*LaTeX* LaTeX is a free typesetting system used to produce PDF reports in IRIS. We recommend MiKTeX, available from [www.miktex.org](http://www.miktex.org).

### Components not needed

Some components were needed in the past but are not any longer.

*X12-ARIMA* Courtesy of the U.S. Census Bureau, the X12-ARIMA program is now incorporated in, and distributed with IRIS. You don't need to care about anything to be able to use it.

*Symbolic Math Toolbox* IRIS is now equipped with its own symbolic/automatic differentiator, so you do not need to have the Symbolic Math Toolbox installed to be able to compute exact Taylor expansions.

### Installing IRIS

*Step 1* Download the latest IRIS zip archive, `IRIS_Tbx_#_YYYYMMDD.zip`, from [www.iris-toolbox.com](http://www.iris-toolbox.com), and save it in a temporary location on your disk.

*Step 2* Unzip the archive into a folder on your hard drive, e.g. `C:\IRIS_Tbx`. We will call this directory the IRIS root folder.

Installing IRIS on a network drive may cause some minor problems, especially on MS Windows systems; check out `help changeNotification` in Matlab.



## Installing IRIS

*Step 3* After installing a new version of IRIS, we recommend that you remove all older versions of IRIS from the Matlab search path, and restart Matlab.

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

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

where C:\IRIS\_Tbx needs to be, obviously, replaced with the proper IRIS root folder chosen in Step 2 above.

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

See also the section on [Starting and quitting IRIS](#) P10.

### Syntax highlighting

You can get the model files (i.e. the files that describe the model equations, variables, parameters) syntax-highlighted. Syntax highlighting improves enormously the readability of the files. It helps you understand the model better, and discover typos and mistakes more quickly.

Add any number of extensions you want to use for model files (such as 'mod', 'model', etc.) to the Matlab editor. Open the menu File - Preferences, and click on the Editor/Debugger - Language tab (make sure 'Matlab' is selected at the top as the Language). Use the Add button in the File extensions panel to associate any number of new extensions with the editor. Re-start the editor. The IRIS model files will be syntax highlighted from that moment on.

## 2 Starting, quitting, and configuring IRIS

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

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

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

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

### Starting and quitting IRIS

- `irisstartup` P16 - Start an IRIS session.
- `irisfinish` P11 - Close the current IRIS session.
- `iriscleanup` P11 - Remove IRIS from Matlab and clean up.

### Getting information about IRIS

- `irisget` P12 - Query current IRIS configuration.
- `irisroot` P14 - Current IRIS root folder.
- `irisrequired` P13 - Throw error if the installed version of IRIS fails to comply with the required minimum.
- `irisversion` P18 - Current IRIS version.

### Changes in configuration

- `irisset` P15 - Change IRIS configuration options.
- `irisreset` P14 - Reset IRIS configuration options to start-up values.
- `irisuserconfig` P17 - User configuration file called at the IRIS start-up.

## Getting on-line help on configuration functions

```
help config  
help function_name
```

---

## ■ **iriscleanup**

Remove IRIS from Matlab and clean up

### Syntax

```
iriscleanup
```

### Description

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

---

## ■ **irisfinish**

Close the current IRIS session

### Syntax

```
irisfinish  
irisfinish -shutup
```

### Description

This function removes all IRIS subfolders from the temporary Matlab search path, and clears persistent variables in some of the backend functions. A short message is displayed with the list of

subfolders removed from the path unless you call use the option `-shutup`. Note that the IRIS root folder stays on the permanent Matlab path.

## Example

---

## ■ `irisget`

Query current IRIS configuration

### Syntax

```
VALUE = irisget(OPTION)
S = irisget()
```

### Input arguments

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

### Output arguments

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

### Description

You can view any of the modifiable options listed in [`irisset`](#) P15, plus the following non-modifiable ones (these cannot be changed by the user):

- `'userConfigPath='` [ char ] - The path to the user configuration file called by the last executed [`irisstartup`](#) P16.
- `'irisRoot='` [ char ] - The current IRIS root directory.
- `'version='` [ char ] - The current IRIS version string.

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

### Example

```
irisget('dateformat')
```

```
ans =
```

```
YFP
```

```
g = irisget();  
g.dateformat
```

```
ans =
```

```
YFP
```

---

## ■ irisrequired

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

### Syntax

```
irisrequired(V)
```

### Input arguments

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

### Description

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

### Example

All of the three calls are valid:

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

---

## ■ irisreset

Reset IRIS configuration options to start-up values

### Syntax

```
irisreset
```

### Description

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

---

## ■ irisroot

Current IRIS root folder

### Syntax

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

### Output arguments

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

### Description

The `irisroot` function is equivalent to the following call to `irisget` P12

```
iriset('irisroot')
```

---

## ■ **iriset**

### Change IRIS configuration options

#### Syntax

```
iriset(OPTION,VALUE)
iriset(OPTION,VALUE,OPTION,VALUE,...)
```

#### Input arguments

- `OPTION` [ char ] - Name of the IRIS configuration option that will be modified.
- `VALUE` [ ... ] - New value that will be assigned to the option.

#### Modifiable IRIS configuration options

##### *Dates and formats*

- `'dateFormat='` [ char | `'YPF'` ] - Date format used to display dates in the command window, CSV databases, and reports. Note that the default date format for graphs is controlled by the `'plotdateFormat='` option. The default `'YPF'` means that the year, frequency letter, and period is displayed. See also help on [dat2str](#) `P231` for more date formatting details. The `'dateFormat='` option is also found in many IRIS functions whenever it is relevant, and can be used to overwrite the `'iriset='` settings.
- `'freqLetters='` [ char | `'YHQBm'` ] - Five letters used to represent the five possible frequencies of the IRIS dates: yearly, half-yearly, quarterly, bi-monthly, and monthly, such as the `'Q'` in `'2010Q1'` denoting.
- `'months='` [ cellstr | {`'January',...,'December'`} ] - Twelve strings representing the names of the twelve months. This option can be used whenever you want to replace the default English names with your local language. .
- `'plotDateFormat='` [ char | `'Y:P'` ] - Date format used to display dates in graphs including graphs in reports. The default is `'Y:P'`, which is different from the `'dateFormat='` option.

## Starting, quitting, and configuring IRIS: `irisstartup`

- `'tseriesFormat='` [ char | empty ] - Format string for displaying time series data on the screen. See help on the Matlab `sprintf` function for how to set up format strings. If empty the default format of the `num2str` function is used.
- `'standinMonth='` [ 'first' | 'last' | numeric ] - This option specifies which month will be used to represent lower-frequency periods (such as a quarters) when a month-displaying format is used in `'dateformat='`.

### *External tools used by IRIS*

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

### *Other options*

- `'extensions='` [ cellstr | {'model','s','q'} ] - List of extensions that are automatically associated with the Matlab editor. The advantage of such associations is that the IRIS [model files](#) [P23](#), [steady-state files](#) [P161](#), and [quick-report](#) [P372](#) files get syntax-highlighted.

## Description

## Example

---

## ■ `irisstartup`

Start an IRIS session

### Syntax

```
irisstartup
irisstartup -shutup
```



## Description

We recommend that you keep the IRIS root directory on the permanent Matlab search path. Each time you wish to start working with IRIS, you run `irisstartup` from the command line. At the end of the session, you can run `irisfinish` [P11](#) to remove IRIS subfolders from the temporary Matlab search path, and to clear persistent variables in some of the backend functions.

The `irisstartup` [P16](#) performs the following steps:

- Adds necessary IRIS subdirectories to the temporary Matlab search path.
- Removes redundant IRIS folders (e.g. other or older installations) from the Matlab search path.
- Resets IRIS configuration options to default, updates the location of TeX/LaTeX executables, and calls `irisuserconfig` [P17](#) to modify the configuration option.
- Associates the default IRIS extensions with the Matlab Editor. If they had not been associated before, Matlab must be re-started for the association to take effect.
- Prints an introductory message on the screen unless `irisstartup` is called with the `-shutup` input argument.

---

## ■ irisuserconfig

User configuration file called at the IRIS start-up

### Syntax

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

### Description

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

The `irisuserconfig.m` file must be an m-file function taking one input argument (a struct with the factory settings), and returning one output argument (a struct with the user-modified settings); see [irisset](#) [P15] for the list of options you can change. In addition, you can also add your own new options, which will be then also accessible through [irisset](#) [P15] and [irisget](#) [P12].

### Example

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

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

This modification will take effect after you next run [irisstartup](#) [P16]. Your graphs will be then fluent in Finnish:

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

---

## ■ irisversion

Current IRIS version

### Syntax

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

### Output arguments

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

## Description

The version string consists of the generation number, followed by a dot and the distribution date (yyyymmdd).

The irisversion function is equivalent to the following call to `irisget` P12

```
irisget('version')
```

### 3 Getting on-line help

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

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

The following help topics are available:

```
help dates
help dates/function_name
help dbase
help dbase/function_name
help modellang
help modellang/keyword
help model
help model/function_name
help plan
help plan/function_name
help poster
help poster/function_name
help logdist
help logdist/function-name
help sstatelang
help sstatelang/keyword
help sstate
help sstate/function_name
help tseries
help tseries/function_name
help VAR
help VAR/function_name
help SVAR
help SVAR/function_name
help BVAR
help BVAR/function_name
help FAVAR
help FAVAR/function_name
help report
help report/function_name
help qreportlang
help qreportlang/keyword
```

## Getting on-line help

```
help qreport  
help qreport/keyword  
help grfun  
help grfun/function_name
```

You can use idoc instead of help in the above list.

Part II —  
Modelling

## 4 Model file language

Model file language is used to write model files. The model files are plain text files (saved under any filename with any extension) that describes the model: its equations, variables, parameters, etc. The model file, on the other hand, does not describe what to do with the model. To run the tasks you want to perform with the model, you need first to load the model file into Matlab using the `model` [P107](#) function. This function creates a model object. Then you write your own m-files using Matlab and IRIS functions to perform the desired tasks with the model object.

Why do all the keywords (except pseudofunctions) start with an exclamation point? Why do the comments have the same style as in Matlab? Why do substitutions and steady-state references use the dollar sign? Because this way, you can get the model files syntax-highlighted in the Matlab editor. Syntax highlighting improves enormously the readability of the files, and helps understand the model more quickly. See [the setup instructions](#) [P8](#) for more details.

### Variables, parameters, substitutions and functions

- `!transition_variables` [P60](#) - List of transition variables.
- `!transition_shocks` [P59](#) - List of transition shocks.
- `!measurement_variables` [P46](#) - List of measurement variables.
- `!measurement_shocks` [P45](#) - List of measurement shocks.
- `!parameters` [P51](#) - List of parameters.
- `!userdiff` [P62](#) - List of user m-file functions that return user-supplied derivatives.

### Equations

- `!transition_equations` [P58](#) - Block of transition equations.
- `!measurement_equations` [P44](#) - Block of measurement equations.
- `!dtrends` [P31](#) - Block of deterministic trend equations.
- `!links` [P41](#) - Define dynamic links.

### Linearised and log-linearised variables

- `!log_variables` [P42](#) - List of log-linearised variables.
- `!allbut` [P26](#) - Inverse list of log-linearised variables.
- `<...>` [P53](#) - Regular expression in log-variable list.

## Model pseudofunctions

Pseudofunctions do not start with an exclamation point.

- `min` [P47] - Define the loss function in a time-consistent optimal policy model.

## Special operators

- `!ttrend` [P61] - Linear time trend in deterministic trend equations.
- `{...}` [P40] - Lag or lead.
- `&` [P53] - Reference to the steady-state level of a variable.
- `!!` [P54] - Steady-state version of an equation.
- `=#` [P32] - Mark an equation for exact non-linear simulation.

## Preparser pseudofunctions

Pseudofunctions do not start with an exclamation point.

- `diff` [P28] - First difference pseudofunction.
- `dot` [P30] - Gross rate of growth pseudofunction.
- `difflog` [P29] - First log-difference pseudofunction.
- `movavg` [P48] - Moving average pseudofunction.
- `movprod` [P49] - Moving product pseudofunction.
- `movsum` [P50] - Moving sum pseudofunction.

## Preparser control commands

- `!substitutions` [P55] - Define text substitutions.
- `!import` [P39] - Include the content of another model file.
- `!export` [P32] - Create a carry-around file to be saved on the disk.
- `!if...!elseif...!else...!end` [P37] - Choose block of code depending on a condition.
- `!switch...!case...!end` [P56] - Switch among several branches of the model code depending on the value of an expression.



- `!for...!do...!end` [P33](#) - For loop for automated creation of model code.
- `%` [P40](#) - Line comments.
- `%{...%}` [P27](#) - Block comments.

### Getting on-line help on model file language

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

```
help modellang
help modellang/!keyword
help modellang/!command
help modellang/pseudofunction
```

### Matlab functions and user functions in model files

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

In your own m-file functions, you can also (optionally) supply the first derivatives that will be used to compute Taylor expansions when the model is being solved, or also the second derivatives that will be used when the function occurs in a loss function. The function must comply with the following requirements. First, you must place the function name under the heading `!userdiff` in the model file. Second, write your function so that it can be called with two extra input arguments on top of the regular input arguments. The first extra input argument is a text string `'diff'` (indicating the call to the function is supposed to return a derivative). The second extra input argument is a number or a vector of two numbers; it determines with respect to which input argument or arguments the first derivative or the second derivative is requested.

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

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

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

Model file language: !allbut

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

returns the second derivative wrt to the first and second input arguments. Note that second derivatives are only needed for functions that occur in an equation defining optimal policy objective, [min](#) P47.

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

### Basic rules IRIS model files

- There can be four types of equations in IRIS models: transition equations which are simply the endogenous dynamic equations, measurement equations which link the model to observables, deterministic trend equations which can be added at the top of measurement equations, and dynamic links which can be used to link some parameters or steady-state values to each other.
- There can be two types of variables and two types of shocks in IRIS models: transition variables and shocks, and measurement variables and shocks.
- Each model must have at least one transition (aka endogenous) variable and one transition equation.
- Each variable, shock, or parameter must be declared in the appropriate declaration section.
- The declaration sections and equations sections can be written in any order.
- You can have as many declaration sections or equations sections of the same kind as you wish in one model file; they all get combined together at the time the model is being loaded.
- The transition variables can occur with lags and leads in transition equations. The transition variables cannot, though, have leads in measurement equations.
- The measurement variables and the shocks cannot have any lags or leads.
- The transition shocks cannot occur in measurement equations, and the measurement shocks cannot occur in transition equations.
- You can choose between linearisation and log-linearisation for each individual transition and measurement variable. Shocks are always linearised.

---

## ■ !allbut

Inverse list of log-linearised variables

## Syntax

```
!log_variables
  !allbut
  VARIABLE_NAME, VARIABLE_NAME,
  VARIABLE_NAME, ...
```

## Description

See help on `!log_variables` [P42](#).

---

## ■ %{...%}

Block comments

## Syntax

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

## Description

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

## Example

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

---

## ■ diff

First difference pseudofunction

### Syntax

```
diff(EXPR)
diff(EXPR,K)
```

### Description

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

$$((\text{EXPR})-(\text{EXPR}\{-1\}))$$

If the input argument K is specified, it expands to

$$((\text{EXPR})-(\text{EXPR}\{K\}))$$

The two derived expressions,  $\text{EXPR}\{-1\}$  and  $\text{EXPR}\{K\}$ , are based on  $\text{EXPR}$ , and have all its time subscripts shifted by  $-1$  or by  $K$  periods, respectively.

### Example

These two lines

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

will expand to

$$\begin{aligned} &((Z)-(Z\{-1\})) \\ &((\log(X\{1\})-\log(Y\{-1\}))-(\log(X\{-1\})-\log(Y\{-3\}))) \end{aligned}$$

## ■ **difflog**

First log-difference pseudofunction

### **Syntax**

```
difflog(EXPR)
difflog(EXPR,K)
```

### **Description**

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

$$(\log(EXPR) - \log(EXPR\{-1\}))$$

If the input argument K is specified, it expands to

$$(\log(EXPR) - \log(EXPR\{K\}))$$

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

### **Example**

The following two lines of code

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

will expand to

$$\begin{aligned} &(\log(Z) - \log(Z\{-1\})) \\ &(\log(X\{1\}/Y\{-1\}) - \log(X\{-1\}/Y\{-3\})) \end{aligned}$$

## ■ dot

Gross rate of growth pseudofunction

### Syntax

```
dot(EXPR)
dot(EXPR,K)
```

### Description

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

$$((\text{expression})/(\text{expression}\{-1\}))$$

If the input argument  $k$  is specified, it expands to

$$((\text{expression})/(\text{expression}\{k\}))$$

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

### Example

The following two lines

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

will expand to

$$\begin{aligned} &((Z)/(Z\{-1\})) \\ &((X+Y)/(X\{-2\}+Y\{-2\})) \end{aligned}$$

## ■ !dtrends

Block of deterministic trend equations

### Syntax for linearised measurement variables

```
!dtrends
  VARIABLE_NAME += EXPRESSION;
  VARIABLE_NAME += EXPRESSION;
  VARIABLE_NAME += EXPRESSION;
  ...
```

### Syntax for log-linearised measurement variables

```
!dtrends
  log(VARIABLE_NAME) += EXPRESSION;
  log(VARIABLE_NAME) += EXPRESSION;
  log(VARIABLE_NAME) += EXPRESSION;
  ...
```

### Syntax with equation labels

```
!dtrends
  'Equation label' VARIABLE_NAME += EXPRESSION;
  'Equation label' LOG(VARIABLE_NAME) += EXPRESSION;
```

## Description

### Example

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

## ■ =#

Mark an equation for exact non-linear simulation

### Syntax

```
LHS =# RHS;
```

### Description

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

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

---

## ■ !export

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

### Syntax

```
!export(FILENAME)
    FILE_CONTENTS
!end
```

### Description

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

The file or files are created and save under the name specified in the !export keyword at the time you load the model using the function [model](#) [P107](#). The contents of the export files is are also stored in the model objects. You can manually re-create and re-save the files by running the function [export](#) [P81](#).



Model file language: `!for...!do...!end`

Note that if no filename is provided or FILENAME is empty, the corresponding `!export` block is discarded without an error or warning.

---

## ■ `!for...!do...!end`

For loop for automated creation of model code

### Short-cut syntax

```
!for
    list_of_tokens
!do
    template
!end
```

### Full syntax

```
!for
    ?control_name = list_of_tokens
!do
    template
!end
```

### Description

Use the `!for...!do...!end` command to specify a template and let the IRIS preparer automatically create multiple instances of the template by iterating over a list of tokens. The preparer cycles over the individual strings from the list; in each iteration, the current string is used to replace all occurrences of the control variable in the template. The name of the control name is either a question mark, `'?'`, in the abbreviated syntax, or any string (not to blank spaces) specified by the user starting with a question mark in the full syntax, such as `'?x'`, `'?#'`, `'?NAME'`, etc.

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

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

Model file language: !for...!do...!end

- all occurrences in the template of 'lower' followed immediately by the control variable, such as 'lower?' in the abbreviated syntax, or 'lower?x', 'lower?#', 'lower?NAME', etc., in the full syntax, are replaced with the currently processed string converted to lowercase.
- all occurrences in the template of 'upper' followed immediately by the control variable are replaced with the currently processed string converted to uppercase.

The list of tokens can be based on Matlab expressions. The expressions must be enclosed in square brackets, and must evaluate into either a numeric vector, a char vector, or a cell array of numerics and/or strings.

### Example 1

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

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

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

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

### Example 2

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

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

Model file language: !for...!do...!end

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

The preparser expands this structure to the following :

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

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

### Example 3

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

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

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

```
!for
```

Model file language: !for...!do...!end

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

#### Example 4

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

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

The preparser produces the following six equations:

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

#### Example 5

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

```
!for
  1, 2, 3, 4, 5, 6, 7
```

Model file language: !if...!elseif...!else...!end

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

can be simplified as follow:

```
!for
    [ 1 : 7 ]
!do
    a? = a?{-1} + res_a?;
!end
```

---

## ■ !if...!elseif...!else...!end

Choose block of code depending on a condition

Syntax with else and elseif clauses

```
!if condition1
    block1
!elseif condition2
    block2
!elseif condition3
    ...
!else
    block3
!end
```

Syntax with an else clause only

```
!if condition1
    block1
!else
    block2
!end
```

### Syntax without an else clause

```
!if condition
    block1
!end
```

### Description

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

Use the !if...!else...!end command to create branches or versions of the model code. Whether a block of code in a particular branch is used or discarded, depends on the condition after the opening !if command and the conditions after subsequent !elseif commands if present. The condition must be a Matlab expression that evaluates to true or false. The condition can refer to model parameters, or to other fields included in the database passed in through the option 'assign=' in the [model](#) P107 function.

### Example 1

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

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

### Example 2

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

Model file language: !import

!end

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

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

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

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

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

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

---

## ■ !import

Include the content of another model file

### Syntax

```
!import(FILENAME)
```

## Description

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

## Example

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

---

## ■ {...}

Lag or lead

## Syntax

```
VARIABLE_NAME{-lag}  
VARIABLE_NAME{lead}  
VARIABLE_NAME{+lead}
```

## Description

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

## Example

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

---

## ■ %

Line comments



## Syntax

% Anything between the percent sign and the line break is discarded.

## Description

## Example

---

# ■ !links

Define dynamic links

## Syntax

```
!links
  PARAMETER_NAME := EXPR;
  VARIABLE_NAME  := EXPR;
```

## Syntax with equation labels

```
!links
  'Equation label' PARAMETER_NAME := EXPR;
  'Equation label' VARIABLE_NAME  := EXPR;
```

## Description

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

The links are automatically refreshed in [solve](#) [P120](#), [sstate](#) [P124](#), and [chksstate](#) [P71](#) functions, and also in each iteration within the [estimate](#) [P76](#) function. They can also be refreshed manually by calling [refresh](#) [P111](#).

The links must not involve parameters occurring in [!dtrends](#) [P31](#) equations that will be estimated using the 'outoflik=' option of the [estimate](#) [P76](#) function.

### Example

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

---

## ■ !log\_variables

List of log-linearised variables

### Syntax

```
!log_variables
  VARIABLE_NAME, VARIABLE_NAME,
  VARIABLE_NAME, ...
```

### Inverted syntax

```
!log_variables
  !allbut
  VARIABLE_NAME, VARIABLE_NAME,
  VARIABLE_NAME, ...
```

### Syntax with regular expression(s)

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

## Description

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

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

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

## Example 1

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

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

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

You can do the same job by writing

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

!log_variables
    !allbut
    r, pie
```

## Example 2

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

```
!transition_variables
```

Model file language: !measurement\_equations

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

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

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

---

## ■ !measurement \_ equations

Block of measurement equations

### Syntax

```
!measurement_equations  
  equation;  
  equation;  
  equation;  
  ...
```

### Syntax with equation labels

```
!measurement_equations  
  equation;  
  'Equation label' equation;  
  equation;  
  ...
```

### Description

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

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

### Example

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

---

## ■ !measurement\_shocks

List of measurement shocks

### Syntax

```
!measurement_shocks
  SHOCK_NAME, SHOCK_NAME, ...
  ...
```

### Syntax with descriptors

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

### Description

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

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

### Example

```
!measurement_shocks
```

Model file language: !measurement\_variables

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

---

## ■ !measurement\_variables

List of measurement variables

### Syntax

```
!measurement_variables  
    variable_name, variable_name, ...  
    ...
```

### Syntax with descriptors

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

### Syntax with steady-state values

```
!measurement_variables  
    variable_name, variable_name, ...  
    variable_name = value
```

### Description

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

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

For each individual variable in a non-linear model, you can also decide if it is to be linearised or log-linearised by listing its name in the `!log_variables` [P42](#) section.

### Example

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

---

## ■ min

Define the loss function in a time-consistent optimal policy model

### Syntax

```
min(DISC) EXPRESSION;
```

### Syntax for exact non-linear simulations

```
min#(DISC) EXPRESSION;
```

### Description

The loss function must be types as one of the transition equations. The DISC is a parameter or an expression defining the discount factor (applied to future dates), the EXPRESSION defines the loss function proper.

If you use the min#(DISC) syntax, all equations created by differentiating the lagrangian w.r.t. individual variables will be earmarked for exact non-linear simulations provided the respective derivative is nonzero.

## Example

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

```
!transition_variables
    x, pi

!transition_shocks
    u

!parameters
    alpha, beta, gamma

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

---

## ■ movavg

Moving average pseudofunction

### Syntax

```
movavg(EXPR)
movavg(EXPR,K)
```

### Description

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

$$(((EXPR)+(EXPR\{-1\})+ \dots +(EXPR\{-(K-1)\}))/-K)$$

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

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



$$(((\text{EXPR})+(\text{EXPR}\{1\})+ \dots +(\text{EXPR}\{K-1\}))/K)$$

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

### Example

The following three lines

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

will expand to

$$\begin{aligned} &(((Z)+(Z\{-1\})+(Z\{-2\})+(Z\{-3\}))/4) \\ &(((Z)+(Z\{-1\})+(Z\{-2\}))/3) \\ &(((X+Y\{-1\})+(X\{1\}+Y))/2) \end{aligned}$$

---

## ■ movsum

Moving product pseudofunction

### Syntax

```
movprod(EXPR)
movprod(EXPR,K)
```

### Description

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

$$((\text{EXPR}) * (\text{EXPR}\{-1\}) * \dots * (\text{EXPR}\{-(K-1)\}))$$

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

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

$$((\text{EXPR}) * (\text{EXPR}\{1\}) * \dots * (\text{EXPR}\{K-1\}))$$

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

### Example

The following two lines

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

will expand to

$$\begin{aligned} &((Z) * (Z\{-1\}) * (Z\{-2\}) * (Z\{-3\})) \\ &((Z) * (Z\{-1\}) * (Z\{-2\})) \\ &((X+Y\{-1\}) * (X\{1\}+Y)) \end{aligned}$$


---

## ■ movsum

Moving sum pseudofunction

### Syntax

```
movsum(EXPR)
movsum(EXPR,K)
```

## Description

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

$$((\text{EXPR})+(\text{EXPR}\{-1\})+ \dots +(\text{EXPR}\{-(K-1)\}))$$

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

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

$$((\text{EXPR})+(\text{EXPR}\{1\})+ \dots +(\text{EXPR}\{K-1\}))$$

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

## Example

The following three lines

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

will expand to

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

---

## ■ !parameters

List of parameters

## Syntax

```
!parameters
  parameter_name, parameter_name, ...
  ...
```

## Syntax with descriptors

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

## Syntax with steady-state values

```
!parameters
  parameter_name, parameter_name, ...
  parameter_name = value
```

## Description

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

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

## Example

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

---

## ■ <...>

Regular expression in log-variable list

### Syntax

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

### Description

See help on [!log\\_variables](#) P42.

---

## ■ &

Reference to the steady-state level of a variable

### Syntax

```
&VARIABLE_NAME
$VARIABLE_NAME
```

### Description

Use either a & or \$ sign in front of a variable name to create a reference to that variable's steady-state level in transition or measurement equations. The two signs, & and \$, are interchangeable.

The steady-state reference will be replaced

- with the variable itself at the time model's steady state is being calculated, i.e. when calling the function [sstate](#) P124;
- with the actually assigned steady-state value at the time the model is being solved, i.e. when calling the function ['solve'](#) P120.

### Example

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

---

## ■ !!

Steady-state version of an equation

### Syntax

```
EQUATION !! STEADY_STATE_EQUATION;
```

### Description

For each transition or measurement equation, you can provide a separate steady-state version. The steady-state version is used when you run the [sstate](#) [P124](#) function. This is useful when you can substantially simplify some parts of the dynamic equations, and help therefore the numerical solver to achieve faster and more accurate results.

Why is it the double exclamation point, !!, that is used to start the steady-state versions of equations? Because if you associate your model file extension with the Matlab editor, anything after an exclamation point is displayed red making it easier to spot the steady-state equations.

### Example 1

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

### Example 2

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

---

## ■ !substitutions

Define text substitutions

### Syntax

```
!substitutions
  SUBSTITUTION_NAME := TEXT_STRING;
  SUBSTITUTION_NAME := TEXT_STRING;
  ...
```

### Description

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

The substitutions can be used in any of the model equations, i.e. in [transition equations](#) [P58], [measurement equations](#) [P44], [deterministic trend equations](#) [P31], and [dynamic links](#) [P41]. Each occurrence of the name of a substitution enclosed in dollar signs, i.e. `$substitution_name$`, in model equations will be replaced with the text string from the substitution's definition.

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

### Example

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

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

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

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

Model file language: `!switch...!case...!otherwise...!end`

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

---

## ■ `!switch...!case...!otherwise...!end`

Switch among several branches of the model code depending on the value of an expression

### Syntax with an otherwise clause

```
!switch expression
  !case value1
    block1
  !case value2
    block2
  ...
  !otherwise
    otherblock
!end
```

### Syntax without an otherwise clause

```
!switch expression
  !case value1
    block1
  !case value2
    block2
  ...
!end
```

### Description

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

Use the `!switch...!case...!end` command to create a larger number of branches of the model



Model file language: !switch...!case...!otherwise...!end

code. Which block of code is actually read in and which blocks are discarded depends on which value in the !case clauses matches the value of the !switch expression. This works exactly as the switch...case...end command in Matlab. The expression after the !switch part of the command must be a valid Matlab expression, and can refer to the model parameters, or to other fields included in the parameter database passed in when you run the `model` [P107](#) function; see [the option 'assign='](#) [P107](#).

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

### Example

```
!switch policy_regime

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

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

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

!end
```

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

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

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

database.

---

## ■ !transition\_equations

Block of transition equations

### Syntax

```
!transition_equations
    equation;
    equation;
    equation;
    ...
```

### Short-cut syntax

```
!equations
    equation;
    equation;
    equation;
    ...
```

### Syntax with equation labels

```
!transition_equations
    equation;
    'Equation label' equation;
    equation;
    ...
```

### Description

The !transition\_equations keyword starts a new block of transition equations (i.e. endogenous equations); the equations can stretch over multiple lines and must be separated by semi-colons.

Model file language: !transition\_shocks

You can have as many equation blocks as you wish in any order in your model file: They all get combined together when you read the model file in.

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

### Example

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

---

## ■ !transition\_shocks

List of transition shocks

### Syntax

```
!transition_shocks
  shock_name, shock_name, ...
  ...
```

### Short-cut syntax

```
!shocks
  shock_name, shock_name, ...
  ...
```

### Syntax with descriptors

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

## Description

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

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

## Example

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

---

## ■ !transition\_variables

List of transition variables

### Syntax

```
!transition_variables
  variable_name, variable_name, ...
  ...
```

### Short-cut syntax

```
!variables
  variable_name, variable_name, ...
  ...
```

### Syntax with descriptors

```
!transition_variables
```

```
variable_name, variable_name, ...  
'Description of the variable...' variable_name
```

### Syntax with steady-state values

```
!transition_variables  
  variable_name, variable_name, ...  
  variable_name = value
```

### Description

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

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

For each individual variable in a non-linear model, you can also decide if it is to be linearised or log-linearised by listing its name in the `!log_variables` [P42](#) section.

### Example

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

---

## ■ !ttrend

Linear time trend in deterministic trend equations

### Syntax

```
!ttrend
```

### Description

#### Example

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

---

## ■ !userdiff

List of user m-file functions that return user-supplied derivatives

### Syntax

```
!userdiff
    FUNCTION_NAME, FUNCTION_NAME,
    FUNCTION_NAME
```

### Description

You can use any functions or your own m-file functions (provided they are visible on the Matlab search path or in the current working directory) in model files. When computing the Taylor expansion of the model equations, IRIS uses symbolic/automatic differentiator for all elementary functions. Other functions are differentiated numerically.

Instead of that, you can supply first derivatives (and also second derivatives in case one of a function occurring in a loss function, [min](#) [P47](#)) for you m-file functions use in the model file. The function must be designed to comply with certain rules, see [Matlab functions and user functions in model files](#) [P23](#), and in addition, must be also declared in the model file itself under the heading !userdiff.

#### Example

## 5 Model objects and functions

Model objects are created by loading a [model file](#) [P23]. Once a model object exists, you can use model functions and standard Matlab functions to write your own m-files to perform the desired tasks, such as calibrate or estimate the model, find its steady state, solve and simulate it, produce forecasts, analyse its properties, and so on.

Model methods:

### Constructor

- [model](#) [P107] - Create new model object based on model file.

### Getting information about models

- [comment](#) [P72] - Get or set user comments in an IRIS object.
- [eig](#) [P75] - Eigenvalues of the model transition matrix.
- [findeqtn](#) [P86] - Find equations by the labels.
- [findname](#) [P87] - Find names of variables, shocks, or parameters by their descriptors.
- [get](#) [P90] - Query model object properties.
- [isstationary](#) [P101] - True if model or specified combination of variables is stationary.
- [islinear](#) [P98] - True for models declared as linear.
- [islog](#) [P98] - True for log-linearised variables.
- [isnan](#) [P100] - Check for NaNs in model object.
- [isname](#) [P99] - True if a string is a model's variable, shock or parameter name.
- [issolved](#) [P100] - True if a model solution exists.
- [length](#) [P103] - Number of alternative parameterisations.
- [omega](#) [P110] - Get or set the covariance matrix of shocks.
- [sspace](#) [P123] - State-space matrices describing the model solution.
- [system](#) [P131] - System matrices before model is solved.
- [userdata](#) [P132] - Get or set user data in an IRIS object.

## Referencing model objects

- `subsasgn` [P128](#) - Subscripted assignment for model and syeq objects.
- `subsref` [P130](#) - Subscripted reference for model and syeq objects.

## Changing model objects

- `alter` [P68](#) - Expand or reduce number of alternative parameterisations.
- `assign` [P69](#) - Assign parameters, steady states, std deviations or cross-correlations.
- `export` [P81](#) - Save carry-around files on the disk.
- `horzcat` [P95](#) - Combine two compatible model objects in one object with multiple parameterisations.
- `refresh` [P111](#) - Refresh dynamic links.
- `stdscale` [P128](#) - Re-scale all std deviations by the same factor.
- `set` [P115](#) - Set modifiable model object properties.
- `single` [P120](#) - Convert solution matrices to single precision.

## Steady state

- `chksstate` [P71](#) - Check if equations hold for currently assigned steady0state values.
- `sstate` [P124](#) - Compute steady state or balance-growth path of the model.
- `sstatefile` [P127](#) - Create a steady-state file based on the model object's steady-state equations.

## Solution, simulation and forecasting

- `diffsrf` [P74](#) - Differentiate shock response functions w.r.t. specified parameters.
- `expand` [P80](#) - Compute forward expansion of model solution for anticipated shocks.
- `jforecast` [P102](#) - Forecast with judgmental adjustments (conditional forecasts).
- `icrf` [P96](#) - Initial-condition response functions.
- `resample` [P114](#) - Resample from the model implied distribution.
- `reporting` [P113](#) - Run reporting equations.



## Model objects and functions

- [shockplot](#) [P117](#) - Short-cut for running and plotting plain shock simulation.
- [simulate](#) [P118](#) - Simulate model.
- [solve](#) [P120](#) - Calculate first-order accurate solution of the model.
- [srf](#) [P122](#) - Shock response functions.

## Model databases

- [emptydb](#) [P76](#) - Create model-specific database with variables, shocks, and parameters.
- [sstatedb](#) [P126](#) - Create model-specific steady-state or balanced-growth-path database.
- [zerodb](#) [P136](#) - Create model-specific zero-deviation database.

## Stochastic properties

- [acf](#) [P66](#) - Autocovariance and autocorrelation functions for model variables.
- [ifrf](#) [P97](#) - Frequency response function to shocks.
- [fevd](#) [P81](#) - Forecast error variance decomposition for model variables.
- [ffrf](#) [P83](#) - Frequency response of transition variables to measurement variables.
- [fmse](#) [P89](#) - Forecast mean square error matrices.
- [vma](#) [P134](#) - Vector moving average representation of the model.
- [xsf](#) [P135](#) - Power spectrum and spectral density of model variables.

## Identification, estimation and filtering

- [bn](#) [P70](#) - Beveridge-Nelson trends.
- [diffloglik](#) [P73](#) - Approximate gradient and hessian of log-likelihood function.
- [estimate](#) [P76](#) - Estimate model parameters by optimising selected objective function.
- [filter](#) [P84](#) - Kalman smoother and estimator of out-of-likelihood parameters.
- [fisher](#) [P87](#) - Approximate Fisher information matrix in frequency domain.
- [lognormal](#) [P106](#) - Characteristics of log-normal distributions returned by filter of forecast.
- [loglik](#) [P104](#) - Evaluate minus the log-likelihood function in time or frequency domain.

- [neighbourhood](#) P109 - Evaluate the local behaviour of the objective function around the estimated parameter values.
- [regress](#) P112 - Centred population regression for selected model variables.
- [VAR](#) P133 - Population VAR for selected model variables.

### Getting on-line help on model functions

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

---

## ■ acf

Autocovariance and autocorrelation functions for model variables

### Syntax

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

### Input arguments

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

### Output arguments

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

### Options

- `'applyTo='` [ `cellstr` | `char` | `Inf` ] - List of variables to which the `'filter='` will be applied; `Inf` means all variables.

- 'contributions=' [ true | false ] - If true the contributions of individual shocks to ACFs will be computed and stored in the 5th dimension of the C and R matrices.
- 'filter=' [ char | empty ] - Linear filter that is applied to variables specified by 'applyto'.
- 'nFreq=' [ numeric | 256 ] - Number of equally spaced frequencies over which the filter in the option 'filter=' is numerically integrated.
- 'order=' [ numeric | 0 ] - Order up to which ACF will be computed.
- 'output=' [ 'namedmat' | 'numeric' ] - Output matrices C and R will be either namedmat objects or plain numeric arrays; if the option 'select=' is used, 'output=' is always a namedmat object.
- 'select=' [ cellstr | Inf ] - Return the ACF matrices for selected variables only; Inf means all variables.

## Description

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

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

## Example 1

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

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

## Example 2

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

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

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

### Example 3

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

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

---

## ■ alter

Expand or reduce number of alternative parameterisations

### Syntax

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

### Input arguments

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

### Output arguments

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

## Description

## Example

---

## ■ assign

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

### Syntax

```
[M,ASSIGNED] = assign(M,P)
[M,ASSIGNED] = assign(M,NAME,VALUE,NAME,VALUE,...)
[M,ASSIGNED] = assign(M,LIST,VALUES)
```

### Syntax for fast assign

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

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

### Syntax for assigning only steady-state levels

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

### Syntax for assignin only steady-state growth rates

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

### Input arguments

- `M [ model ]` - Model object.
- `P [ struct | model ]` - Database whose fields refer to parameter names, variable names, std deviations, or cross-correlations; or another model object.
- `NAME [ char ]` - A parameter name, variable name, std deviation, cross-correlation, or a regular expression that will be matched against model names.
- `VALUE [ numeric ]` - A value (or a vector of values in case of multiple parameterisations) that will be assigned.
- `LIST [ cellstr ]` - A list of parameter names, variable names, std deviations, or cross-correlations.
- `VALUES [ numeric ]` - A vector of values.

### Output arguments

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

### Description

### Example

---

## ■ bn

### Beveridge-Nelson trends

### Syntax

`D = bn(M,D,RANGE,...)`

### Input arguments

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

### Output arguments

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

### Options

- `'deviations=' [ true | false ]` - Input and output data are deviations from balanced-growth paths.
- `'dtrends=' [ 'auto' | true | false ]` - Measurement variables in input and output data include deterministic trends specified in [!dtrends](#) P31 equations.

### Description

### Example

---

## ■ chksstate

Check if equations hold for currently assigned steady0state values

### Syntax

```
[FLAG,LIST] = chksstate(M,...)
[FLAG,DISCREP,LIST] = chksstate(M,...)
```

### Input arguments

- `M [ model ]` - Model object.

### Output arguments

- FLAG [ true | false ] - True if discrepancy between LHS and RHS is smaller than 'tolerance=' in each equation.
- DISCREP [ numeric ] - Discrepancy between LHS and RHS in each equation evaluated at two consecutive times.
- LIST [ cellstr ] - List of equations in which the discrepancy between LHS and RHS is greater than 'tolerance='.

### Options

- 'error=' [ true | false ] - Throw an error if one or more equations do not hold.
- 'refresh=' [ true | false ] - Refresh dynamic links before evaluating the equations.
- 'sort=' [ true | false ] - Undocumented option.
- 'ssstateEqtn=' [ true | false ] - If false, the dynamic model equations will be checked; if true, the steady-state versions of the equations (wherever available) will be checked.
- 'tolerance=' [ numeric | getrealsmall() ] - Tolerance.
- 'warning=' [ true | false ] - Display warnings produced by this function.

### Description

### Example

---

## ■ comment

Get or set user comments in an IRIS object

### Syntax for getting user comments

```
C = comment(OBJ)
```



### Syntax for assigning user comments

```
OBJ = comment(OBJ,C)
```

### Input arguments

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

### Output arguments

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

### Description

### Example

---

## ■ diffloglik

Approximate gradient and hessian of log-likelihood function

### Syntax

```
[MINUSL,GRAD,HESS,V] = diffloglik(M,D,RANGE,LIST,...)
```

### Input arguments

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

### Output arguments

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

### Options

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

See help on [model/filter](#) P84 for other options available.

### Description

### Example

---

## ■ `diffsrf`

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

### Syntax

```
S = diffsrf(M,RANGE,LIST,...)
S = diffsrf(M,NPER,LIST,...)
```

### Input arguments

- `M` [ model ] - Model object whose response functions will be simulated and differentiated.
- `RANGE` [ numeric ] - Simulation date range with the first date being the shock date.

- `NP` [ numeric ] - Number of simulation periods.
- `LIST` [ char | cellstr ] - List of parameters w.r.t. which the shock response functions will be differentiated.

#### Output arguments

- `S` [ struct ] - Database with shock response derivatives stored in multivariate time series.

#### Options

See [model/srf](#) P122 for options available.

#### Description

#### Example

---

## ■ eig

Eigenvalues of the model transition matrix

#### Syntax

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

#### Input arguments

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

#### Output arguments

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

## Description

## Example

---

### ■ emptydb

Create model-specific database with variables, shocks, and parameters

## Syntax

```
d = emptydb(m)
```

## Input arguments

- `m [ model | bkwmodel ]` - Model or bkwmodel object for which the empty database will be created.

## Output arguments

- `d [ struct ]` - Database with an empty tseries object for each variable and each shock, and an empty array for each parameter.

## Description

## Example

---

### ■ estimate

Estimate model parameters by optimising selected objective function

## Syntax

```
[P,POS,COV,HESS,M,V,~,~,DELTA,PDELTA] = estimate(M,D,RANGE,E,...)
```

## Input arguments

- `M [ struct ]` - Model object.
- `D [ struct | cell ]` - Input database or datapack from which the measurement variables will be taken.
- `RANGE [ struct ]` - Date range.
- `E [ struct ]` - Database with the list of parameters that will be estimated, and the estimation specifications for each of them.

## Output arguments

- `P [ struct ]` - Database with point estimates of requested parameters.
- `POS [ poster ]` - Posterior, [poster](#) P148, object; this object also gives you access to the value of the objective function at optimum or at any point in the parameter space, see the [poster/eval](#) P150 function.
- `COV [ numeric ]` - Approximate covariance matrix for the estimates of parameters with slack bounds based on the asymptotic Fisher information matrix (not on the Hessian returned from the optimisation routine).
- `HESS [ cell ]` - `HESS{1}` is the total hessian of the objective function; `HESS{2}` is the contributions of the priors to the hessian.
- `M [ model ]` - Model object solved with the estimated parameters (including out-of-likelihood parameters and common variance factor).

The remaining five output arguments, `V`, `F`, `PE`, `DELTA`, `PDELTA`, are the same as the [model/loglik](#) P104 output arguments of the same names.

A tilde, `~`, denotes a void (unused) output argument that is included for backward compatibility.

## Options

- `'exclude=' [ cellstr | empty ]` - List of measurement variables that are to be excluded from the likelihood function (treated as deterministic).
- `'filter=' [ cell | empty ]` - Cell array of options that will be passed on to the Kalman filter; see help on [model/filter](#) P84 for the options available.
- `'maxIter=' [ numeric | 500 ]` - Maximum number of iterations allowed.
- `'maxFunEvals=' [ numeric | 2000 ]` - Maximum number of objective function calls allowed.

- 'noSolution=' [ 'error' | 'penalty' ] - Specifies what happens if solution or steady state fails to solve in an iteration: 'error=' stops the execution with an error message, 'penalty=' returns an extremely low value of the likelihood.
- 'objective=' [ '-loglik' | 'prederr' ] - Objective function optimised; it can be minus the log likelihood function or weighted sum of prediction errors, either adjusted for prior distributions.
- 'objectiveSample=' [ numeric | Inf ] - The objective function will be computed on this subrange only; Inf means the entire range will be used.
- 'optimSet=' [ cell | empty ] - Cell array used to create the Optimization Toolbox options structure; work only with 'solver=' 'default'.
- 'refresh=' [ true | false ] - Refresh dynamic links in each iteration.
- 'solve=' [ true | false ] - Re-compute solution in each iteration.
- 'solver=' [ 'default' | cell | function\_handle ] - Minimisation procedure; 'default' means the Optimization Toolbox fminunc or fmincon functions. You can enter a function handle to your own optimisation procedure, or a cell array with a function handle and additional input arguments — see below.
- 'ssstate=' [ true | false | cell | function\_handle ] - Re-compute steady state in each iteration. You can specify a cell array with options for the ssstate function, or a function handle whose behaviour is described below.
- 'tolFun=' [ numeric | 1e-6 ] - Termination tolerance on the objective function.
- 'tolX=' [ numeric | 1e-6 ] - Termination tolerance on the estimated parameters.

## Description

In the input parameter database, E, you can provide the following four specifications for each parameter:

```
E.parameter_name = {start,lower,upper,logprior}
```

where start is the value from which the numerical optimisation will start, lower is the lower bound, upper is the upper bound, and logprior is a function handle expected to return the log of the prior density. You can use the [logdist](#) P155 package to create function handles for some of the basic prior distributions.

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

—User-supplied optimisation (minimisation) routine

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

```
[POPT,OBJOPT,HESS] = yourminfunc(F,P0,PLOW,PHIGH,OPT)
```

with the following input arguments:

- F is a function handle to the function minimised;
- P0 is a 1-by-N vector of initial parameter values;
- PLOW is a 1-by-N vector of lower bounds (with -Inf indicating no lower bound);
- PHIGH is a 1-by-N vector of upper bounds (with Inf indicating no upper bounds);
- OPT is an Optim Tbx style struct with the optimisation settings (tolerance, number of iterations, etc); of course you may simply ignore this information and leave the input argument unused.

and the following output arguments:

- POPT is a 1-by-N vector of estimated parameters;
- OBJOPT is the value of the objective function at optimum;
- HESS is a N-by-N approximate hessian matrix at optimum.

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

```
{@yourminfunc,ARG1,ARG2,...}
```

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

```
[POPT,OBJOPT,HESS] = yourminfunc(F,P0,PLOW,PHIGH,OPT,ARG1,ARG2,...)
```

—User-supplied steady-state solver

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

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

```
[M,SUCCESS] = yoursstatesolver(M)
```

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

## Example

---

## ■ expand

Compute forward expansion of model solution for anticipated shocks

### Syntax

```
m = expand(m,k)
```

### Input arguments

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



### Output arguments

- `m [ model ]` - Model object with the solution expanded.

### Description

### Example

---

## ■ export

Save carry-around files on the disk

### Syntax

```
export(M)
```

### Input arguments

- `M [ model ]` - Model object whose carry-around m-files (written in underlying the model file) will be saved on the disk.

### Description

See the IRIS model language keyword `!export` [P32](#) for help on how to write carry-around m-files in model files.

### Example

---

## ■ fevd

Forecast error variance decomposition for model variables

## Syntax

```
[X,Y,LIST,A,B] = fevd(M,RANGE,...)
[X,Y,LIST,A,B] = fevd(M,NPER,...)
```

## Input arguments

- `M` [ `model` ] - Model object for which the decomposition will be computed.
- `RANGE` [ `numeric` ] - Decomposition date range with the first date beign the first forecast period.
- `NPER` [ `numeric` ] - Number of periods for which the decomposition will be computed.

## Output arguments

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

## Options

- `'output='` [ `'namedmat'` | `numeric` ] - Output matrices `X` and `Y` will be either `namedmat` objects or plain numeric arrays; if the option `'select='` is used, `'output='` is always `'namedmat'`.
- `'select='` [ `char` | `cellstr` ] - Return the decomposition matrices, `X` and `Y`, for selected variables and/or shocks only; `Inf` means all variables. This option does not apply to the database outputs, `A` and `B`.

## Description

## Example

---

## ■ ffrf

Frequency response of transition variables to measurement variables

### Syntax

```
[F,LIST] = ffrf(m,freq,...)
```

### Input arguments

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

### Output arguments

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

### Options

- `'maxIter='` [ numeric | 500 ] - Maximum number of iteration when computing the steady-state Kalman filter.
- `'output='` [ 'namedmat' | numeric ] - Output matrix `F` will be either a `namedmat` object or a plain numeric array; if the option `'select='` is used, `'output='` is always `'namedmat'`.
- `'select='` [ char | cellstr | Inf ] - Return the frequency response function for selected variables only; `Inf` means all variables.
- `'tolerance='` [ numeric | 1e-7 ] - Convergence tolerance when computing the steady-state Kalman filter.

### Description

### Example

## ■ filter

Kalman smoother and estimator of out-of-likelihood parameters

### Syntax

```
[M,OUTP,V,DELTA,PE] = filter(M,INP,RANGE,...)
[M,OUTP,V,DELTA,PE] = filter(M,INP,RANGE,J,...)
```

### Input arguments

- M [ model ] - Solved model object.
- INP [ struct | cell ] - Input database or datapack from which the measurement variables will be taken.
- RANGE [ numeric ] - Filter date range.
- J [ struct ] - Database with tunes on the mean of shocks and/or time-varying std devs of shocks.

### Output arguments

- M [ model ] - Model object with updates of std devs (if 'relative=' is true) and/or updates of out-of-likelihood parameters (if 'outoflik=' is non-empty).
- OUTP [ struct | cell ] - Output struct with smoother or prediction data.
- V [ numeric ] - Estimated variance scale factor if the 'relative=' options is true; otherwise V is 1.
- DELTA [ struct ] - Database with estimates of out-of-likelihood parameters.
- PE [ struct ] - Database with prediction errors for measurement variables.

### Options

- 'ahead=' [ numeric | 1 ] - Predictions will be computed this number of period ahead.
- 'deviation=' [ true | false ] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [ 'auto' | true | false ] - Measurement data contain deterministic trends.

## Model objects and functions: filter

- 'data=' [ 'predict' | 'smooth' | 'predict,smooth' ] - Return smoother data or prediction data or both.
- 'exclude=' [ cellstr | empty ] - List of measurement variables that will be excluded from the likelihood function.
- 'initCond=' [ 'fixed' | 'optimal' | 'stochastic' | struct ] - Method or data to initialise the Kalman filter.
- 'lastSmooth=' [ numeric | Inf ] - Last date up to which to smooth data backward from the end of the range; if Inf smoother will run on the entire range.
- 'meanOnly=' [ true | false ] - Return a plain database with mean data only.
- 'outOfLik=' [ cellstr | empty ] - List of parameters in deterministic trends that will be estimated by concentrating them out of the likelihood function.
- 'objective=' [ '-loglik' | 'prederr' ] - Objective function computed; can be either minus the log likelihood function or weighted sum of prediction errors.
- 'objRange=' [ numeric | Inf ] - The objective function will be computed on this subrange only; Inf means the entire filter range.
- 'precision=' [ 'double' | 'single' ] - Numeric precision to which output data will be stored; all calculations themselves always run to double precision.
- 'rollback=' [ numeric | empty ] - Date up to which to roll back individual observations on measurement variables from the end of the sample.
- 'relative=' [ true | false ] - Std devs of shocks assigned in the model object will be treated as relative std devs, and a common variance scale factor will be estimated.
- 'returnMse=' [ true | false ] - Return MSE matrices for predetermined state variables; these can be used for settin up initial condition in subsequent call to another filter or jforecast.
- 'returnStd=' [ true | false ] - Return database with std devs of model variables.
- 'tolMse=' [ numeric | 0 ] - Tolerance under which two MSE matrices in two consecutive periods will be treated as equal, and the Kalman gain system will be re-used, not re-computed.
- 'weighting=' [ numeric | empty ] - Weighting vector or matrix for prediction errors when 'objective=' 'prederr'; empty means prediction errors are weighted equally.

## Options for models with non-linearised equations

- 'nonlinearise=' [ numeric | 0 ] - If non-zero the prediction step in the Kalman filter will be run in an exact non-linear mode using the same technique as [model/simulate](#) P118.
- 'simulate=' [ cell | empty ] - Options passed in to simulate when invoking the non-linear simulation in the prediction step; only used when nonlinearise= is greater than 0.

## Description

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

## Example

---

## ■ findeqtn

Find equations by the labels

## Syntax

```
[EQTN,EQTN,...] = findeqtn(M,LABEL,LABEL,...)
[LIST,LIST,...] = findeqtn(M,'-rexp',REXP,REXP,...)
```

## Input arguments

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

## Output arguments

- EQTN [ char ] - First equation found with the label LABEL.
- LIST [ cellstr ] - List of equations whose labels match the regular expression REXP.

## Description

## Example

---

## ■ findname

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

### Syntax

```
[NAME,NAME,...] = findname(M,DESC,DESC,...)
[LIST,LIST,...] = findname(M,'-rexp',REXP,REXP,...)
```

### Input arguments

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

### Output arguments

- NAME [ char ] - First name found with the descriptor DESC.
- LIST [ cellstr ] - List of names whose descriptors match the regular expression REXP.

### Description

### Example

---

## ■ fisher

Approximate Fisher information matrix in frequency domain

### Syntax

```
[F,Fi,delta,freq] = fisher(m,nper,list,...)
```

### Input arguments

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

### Output arguments

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

### Options

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

### Description

### Example



## ■ fmse

Forecast mean square error matrices

### Syntax

```
[M,LIST,D] = fmse(M,NPER,...)
[M,LIST,D] = fmse(M,RANGE,...)
```

### Input arguments

- M [ model ] - Model object for which the forecast MSE matrices will be computed.
- NPER [ numeric ] - Number of periods.
- RANGE [ numeric ] - Date range.

### Output arguments

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

### Options

- 'output=' [ 'namedmat' | numeric ] - Output matrix M will be either a namedmat object or a plain numeric array; if the option 'select=' is used, 'output=' is always 'namedmat'.
- 'select=' [ cellstr | Inf ] - Return the FMSE matrices for selected variables only; Inf means all variables. The option does not apply to the database output D.

### Description

### Example

---

## ■ get

Query model object properties

### Syntax

```
ANS = get(M, QUERY)
[ANS, ANS, ...] = get(M, QUERY, QUERY, ...)
```

### Input arguments

- M [ model ] - Model object.
- QUERY [ char ] - Query string.

### Output arguments

- ANS [ ... ] - Answer to the query.

### Valid queries on model objects

Below is the categorised list of model properties and attributes that can be queried/accessed by the get function. Note that letter 'y' is used in various contexts to denote measurement variables or equations, 'x' transition variables or equations, 'e' shocks, 'p' parameters, 'd' deterministic trend equations, 'l' dynamic links, and 'r' reporting equations. The property names are case insensitive.

#### *Steady state*

- 'sstate' — Returns [ struct ] a database with the steady states for all model variables. The steady states are described by complex numbers in which the real part is the level and the imaginary part is the growth rate.
- 'sstateLevel' — Returns [ struct ] a database with the steady-state levels for all model variables.
- 'sstateGrowth' — Returns [ struct ] a database with steady-state growth (first difference for linearised variables, gross rate of growth for log-linearised variables) for all model variables.

- 'dtrends' — Returns [ struct ] a database with the effect of the deterministic trends on the measurement variables. The effect is described by complex numbers the same way as the steady state.
- 'dtrendsLevel' — Returns [ struct ] a database with the effect of the deterministic trends on the steady-state levels of the measurement variables.
- 'dtrendsGrowth' — Returns [ struct ] a database with the effect of deterministic trends on steady-state growth of the measurement variables.
- 'sstate+dtrends' — Returns [ struct ] the same as 'sstate' except that the measurement variables are corrected for the effect of the deterministic trends.
- 'sstateLevel+dtrendsLevel' — Returns [ struct ] the same as 'sstateLevel' except that the measurement variables are corrected for the effect of the deterministic trends.
- 'sstateGrowth+dtrendsGrowth' — Returns [ struct ] the same as 'sstateGrowth' except that the measurement variables are corrected for the effect of the deterministic trends.

*Variables, parameters and equations*

- 'yList' — Returns [ cellstr ] the list of measurement variables in order of their appearance in the model code declarations.
- 'xList' — Returns [ cellstr ] the list of transition variables in order of their appearance in the model code declarations.
- 'eList' — Returns [ cellstr ] the list of shocks in order of their appearance in the model code declarations.
- 'eyList' — Returns [ cellstr ] the list of measurement shocks in order of their appearance in the model code declarations; only those shocks that actually occur in at least one measurement equation are returned.
- 'exList' — Returns [ cellstr ] the list of transition shocks in order of their appearance in the model code declarations; only those shocks that actually occur in at least one transition equation are returned.
- 'pList' — Returns [ cellstr ] the list of the parameter names in order of their appearance in the model code. The list does not include the names of std deviations and cross-correlations.
- 'stdList' — Returns [ cellstr ] the list of the names of the standard deviations for the shocks in order of the appearance of the corresponding shocks in the model code.
- 'corrList' — Returns [ cellstr ] the list of the names of cross-correlation coefficients for the shocks in order of the appearance of the corresponding shocks in the model code.

- 'stdCorrList' — Returns [ cellstr ] the list of the names of std deviations and cross-correlation coefficients for the shocks in order of the appearance of the corresponding shocks in the model code.
- 'yEqtn' — Returns [ cellstr ] the list of the measurement equations in order of their appearance in the model code.
- 'xEqtn' — Returns [ cellstr ] the list of the transition equations in order of their appearance in the model code.
- 'dEqtn' — Returns [ cellstr ] the list of the deterministic-trend equations in order of the appearance of the respective measurement variables in the model code declarations.
- 'lEqtn' — Returns [ cellstr ] the list of the dynamic links in order of the appearance in the model code.
- 'rEqtn' — Returns [ cellstr ] the list of the reporting equations in order of their appearance in the model code.

*First-order Taylor expansion of equations*

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

*Descriptions of variables, parameters, and shocks*

- 'description' — Returns [ struct ] a database with user descriptions of model variables, shocks, and parameters.
- 'yDescription' — Returns [ cellstr ] user descriptions of measurement variables.
- 'xDescription' — Returns [ cellstr ] user descriptions of transition variables.
- 'eDescription' — Returns [ cellstr ] user descriptions of shocks.
- 'pDescription' — Returns [ cellstr ] user descriptions parameters.

### *Labels*

- 'yLabels' — Returns [ cellstr ] user labels added to measurement equations.
- 'xLabels' — Returns [ cellstr ] user labels added to transition equations.
- 'dLabels' — Returns [ cellstr ] user labels added to deterministic-trend equations.
- 'lLabels' — Returns [ cellstr ] user labels added to dynamic links.
- 'rLabels' — Returns [ cellstr ] user labels added to reporting equations.

### *Parameter values*

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

### *Eigenvalues*

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

### *Model structure, solution, build*

- 'build' — Returns [ numeric ] IRIS version number under which the model object has been built.
- 'log' — Returns [ struct ] a database with true for each log-linearised variables, and false for each linearised variable.
- 'maxLag' — Returns [ numeric ] the maximum lag in the model.

- 'maxLead' — Returns [ numeric ] the maximum lead in the model.
- 'stationary' — Returns [ struct ] a database with true for each stationary variables, and false for each unit-root (non-stationary) variables (under current solution).
- 'nonStationary' — Returns [ struct ] a database with true for each unit-root (non-stationary) variable, and false for each stationary variable (under current solution).
- 'stationaryList' — Returns [ cellstr ] the list of stationary variables (under current solution).
- 'nonStationaryList' — Returns [ cellstr ] cell with the list of unit-root (non-stationary) variables (under current solution).
- 'initCond' — Returns [ cellstr ] the list of the lagged transition variables that need to be supplied as initial conditions in simulations and forecasts. The list of the initial conditions is solution-specific as the state-space coefficients at some of the lags may evaluate to zero depending on the current parameters.
- 'yVector' — Returns [ cellstr ] the list of measurement variables in order of their appearance in the rows and columns of state-space matrices (effectively identical to 'yList') from the [model/sspace](#) P123 function.
- 'xVector' — Returns [ cellstr ] the list of transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the [model/sspace](#) P123 function.
- 'xfVector' — Returns [ cellstr ] the list of forward-looking (i.e. non-predetermined) transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the [model/sspace](#) P123 function.
- 'xbVector' — Returns [ cellstr ] the list of backward-looking (i.e. predetermined) transition variables, and their auxiliary lags and leads, in order of their appearance in the rows and columns of state-space matrices from the [model/sspace](#) P123 function.
- 'eVector' — Returns [ cellstr ] the list of the shocks in order of their appearance in the rows and columns of state-space matrices (effectively identical to 'eList') from the [model/sspace](#) P123 function.

## Description

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

The expressions returned by 'derivatives' can, in general, refer to

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

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

### Example

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

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

d{k}

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

w{k}

---

## ■ horzcat

Combine two compatible model objects in one object with multiple parameterisations

### Syntax

```
m = [m1,m2,...]
```

### Input arguments

- `m1, m2 [ model ]` - Compatible model objects that will be combined; the input models must be based on the same model file.

### Output arguments

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

### Description

### Example

---

## ■ icrf

### Initial-condition response functions

### Syntax

```
S = icrf(M,NPER,...)
S = icrf(M,RANGE,...)
```

### Input arguments

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

### Output arguments

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



## Options

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

## Description

## Example

---

## ■ ifrf

Frequency response function to shocks

## Syntax

```
[W,LIST] = ifrf(M,FREQ,...)
```

## Input arguments

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

## Output arguments

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

## Options

- 'output=' [ 'namedmat' | numeric ] - Output matrix W will be either a namedmat object or a plain numeric array; if the option 'select=' is used, 'output=' is always 'namedmat'.

- 'select=' [ char | cellstr | Inf ] - Return the frequency response function only for selected variables and/or selected shocks.

### Description

### Example

---

## ■ islinear

True for models declared as linear

### Syntax

```
flag = islinear(m)
```

### Input arguments

- m [ model ] - Queried model object.

### Output arguments

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

### Description

### Example

---

## ■ islog

True for log-linearised variables

### Syntax

```
flag = islog(m,name)
```

### Input arguments

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

### Output arguments

- `flag` [ true | false ] - True for variables declared as log-linear in a non-linear model.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ isname

True if a string is a model's variable, shock or parameter name

### Syntax

```
[flag,flag,...] = isname(m,name,name,...)
```

### Input arguments

- `m` [ model ] - Model object.
- `name` [ char ] - A text string that will be compared with the model object's names.

### Output arguments

- `flag` [ true | false ] - True if the tested string is the model object's name.

## Description

## Example

---

### ■ isnan

Check for NaNs in model object

## Syntax

```
[FLAG,LIST] = isnan(M,'parameters')  
[FLAG,LIST] = isnan(M,'sstate')  
[FLAG,LIST] = isnan(M,'derivatives')
```

## Input arguments

- M [ model ] - Model object.

## Output arguments

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

## Description

## Example

---

### ■ issolved

True if a model solution exists

### Syntax

```
flag = issolved(m)
```

### Input arguments

- `m [ model ]` - Model object.

### Output arguments

- `flag [ true | false ]` - True for each parameterisation for which a stable unique solution has been found and exists currently in the model object.

### Description

### Example

---

## ■ `isstationary`

True if model or specified combination of variables is stationary

### Syntax

```
FLAG = isstationary(M)  
FLAG = isstationary(M,EXPRESSION)
```

### Input arguments

- `M [ model ]` - Model object.
- `EXPRESSION [ char ]` - Combination of transition variables to be tested.

### Output arguments

- `'flag=' [ true | false ]` - True if the model (if called without a second input argument) or the specified combination of variables (if called with a second input argument) is stationary.

## Description

## Example

---

## ■ jforecast

Forecast with judgmental adjustments (conditional forecasts)

## Syntax

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

## Input arguments

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

## Output arguments

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

## Options

- 'anticipate=' [ true | false ] - If true, real future shocks are anticipated, imaginary are unanticipated; vice versa if false.
- 'currentOnly=' [ true | false ] - If true, MSE matrices will be computed only for current-dated variables, not for their lags or leads.
- 'deviation=' [ true | false ] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [ 'auto' | true | false ] - Measurement data contain deterministic trends.
- 'initCond=' [ 'data' | 'fixed' ] - Use the MSE for the initial conditions if found in the input data or treat the initial conditions as fixed.

- 'meanOnly=' [ true | false ] - Return only mean data, i.e. point estimates.
- 'plan=' [ plan ] - Simulation plan specifying the exogenised variables and endogenised shocks.
- 'vary=' [ struct | empty ] - Database with time-varying std deviations or cross-correlations of shocks.

### Description

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

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

### Description

### Example

---

## ■ length

Number of alternative parameterisations

### Syntax

N = length(M)

### Input arguments

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

### Output arguments

- N [ numeric ] - Number of alternative parameterisations.

## Description

## Example

---

## ■ loglik

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

### Full syntax

```
[OBJ,V,F,PE,DELTA,PDELTA] = loglik(M,D,RANGE,...)
[OBJ,V,F,PE,DELTA,PDELTA,~,PRED,SMOOTH] = loglik(M,D,RANGE,...)
```

### Syntax for fast one-off likelihood evaluation

```
OBJ = loglik(M,D,RANGE,...)
```

### Syntax for repeated fast likelihood evaluations

```
% Initialise
loglik(M,D,RANGE,...,'persist=',true);

% Fast loglik for different parameter sets
M... = ...; % Change parameters.
L = loglik(M); % Evaluate likelihood.
...
M... = ...; % Change parameters.
L = loglik(M); % Evaluated likelihood
...
% etc.
```

### Input arguments

- `M [ model ]` - Model object on which the likelihood of the input data will be evaluated.



- D [ struct | cell ] - Input database or datapack from which the measurement variables will be taken.
- RANGE [ numeric ] - Date range.

## Output arguments

### *Test title*

- OBJ [ numeric ] - Value of minus the log-likelihood function (or other objective function if specified in options).
- V [ numeric ] - Estimated variance scale factor if the 'relative=' options is true; otherwise V is 1.
- F [ numeric ] - Sequence of forecast MSE matrices for measurement variables.
- PE [ struct ] - Database with prediction errors for measurement variables; exp of prediction errors for measurement variables declared as log-variables.
- DELTA [ struct ] - Estimates of deterministic trend parameters specified in through the 'outoflik=' option.
- PDELTA [ numeric ] - MSE matrix of the estimates of the 'outoflik=' parameters.
- PRED [ struct | cell ] - Output database or datapack with the Kalman predictor data.
- SMOOTH [ struct | cell ] - Output database or datapack with the Kalman smoother data.

## Options

- 'persist=' [ true | false ] — Pre-process and store the overhead (data and options) for subsequent fast calls.

See help on [model/filter](#) P84 for other options available.

## Description

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

Not specifying the last two output arguments, PRED and SMOOTH, if you really don't need them will speed up the function significantly because the prediction data will not have to be stored, and the smoother data will not be calculated at all.

The `loglik` function returns identical results as the `model/filter` [P84](#) function, only the output arguments are arranged in different ways.

### Example

---

## ■ lognormal

Characteristics of log-normal distributions returned by filter of forecast

### Syntax

```
d = lognormal(m,d,...)
```

### Input arguments

- `m [ model ]` - Model on which the filter or forecast function has been run.
- `d [ struct ]` - Output structure from the filter or forecast functions.

### Output arguments

- `d [ struct ]` - Output structure with new databases.

### Options

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

## Description

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ model

Create new model object based on model file

### Syntax

```
m = model(fname,...)
m = model(m,...)
```

### Input arguments

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

### Output arguments

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

### Options

- `'multiple='` [ `true` | `false` ] - Allow each variable, shock, or parameter name to be declared (and assigned) more than once in the model file.
- `'assign='` [ `struct` | `empty` ] - Assign model parameters and/or steady states from this database at the time the model objects are being created.
- `'baseYear='` [ `numeric` | `2000` ] - Base year for constructing deterministic time trends.
- `'comment='` [ `char` | `empty` ] - Text comment attached to the model object.
- `'epsilon='` [ `numeric` | `eps^(1/4)` ] - The minimum relative step size for numerical differentiation.
- `'linear='` [ `true` | `false` ] - Indicate linear models.

- 'removeLeads=' [ true | false ] - Remove all leads from the state-space vector, keep included only current dates and lags.
- 'std=' [ numeric | 1 for linear models | 0.01 for non-linear models ] - Default standard deviation for model shocks.
- 'userdata=' [ ... | empty ] - Attach user data to the model object.

## Description

—Loading a model file

The model function can be used to read in a [model file](#) P23 named fname, and create a model object m based on the model file. You can then work with the model object in your own m-files, using using the IRIS [model functions](#) P63 and standard Matlab functions.

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

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

## Example 1

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

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

## Example 2

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

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

Note that this is equivalent to

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

```
m = assign(m,P);
```

unless some of the parameters passed in to the model fuction are needed to evaluate `if` [P37](#) or `!switch` [P56](#) expressions.

---

## ■ neighbourhood

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

### Syntax

```
[D,FIGH,AXH,OBJH,LIKH,ESTH,BH] = neighbourhood(M,PS,NEIGH,...)
```

### Input arguments

- M [ model | bkwmodel ] - Model or bkwmodel object.
- PS [ poster ] - Posterior simulator (poster) object returned by the `model/estimate` [P76](#) function.
- NEIGH [ numeric ] - The neighbourhood in which the objective function will be evaluated defined as multiples of the parameter estimates.

### Output arguments

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

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

- FIGH [ numeric ] - Handles to the figures created.
- AXH [ numeric ] - Handles to the axes objects created.
- OBJH [ numeric ] - Handles to the objective function curves plotted.
- LIKH [ numeric ] - Handles to the data likelihood curves plotted.

- ESTH [ numeric ] - Handles to the actual estimate marks plotted.
- BH [ numeric ] - Handles to the bounds plotted.

### Options

- 'plot=' [ true | false ] - Call the [grfun.plotneigh](#) P382 function from within neighbourhood to visualise the results.
- 'neighbourhood=' [ struct | empty ] - Struct specifying the neighbourhood points for some of the parameters; these points will be used instead of those based on NEIGH.

### Plotting options

See help on [grfun.plotneigh](#) P382 for options available when you choose 'plot=' true.

### Description

In the output database, D, each parameter is a 1-by-3 cell array. The first cell is a vector of parameter values at which the local behaviour was investigated. The second cell is a matrix with two column vectors: the values of the overall minimised objective function (as set up in the [estimate](#) P76 function), and the values of the data likelihood component. The third cell is a vector of four numbers: the parameter estimate, the value of the objective function at optimum, the lower bound and the upper bound.

### Example

---

## ■ omega

Get or set the covariance matrix of shocks

### Syntax for getting covariance matrix

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

### Syntax for setting covariance matrix

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

#### Input arguments

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

#### Output arguments

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

### Description

#### Example

---

## ■ refresh

Refresh dynamic links

### Syntax

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

#### Input arguments

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

### Output arguments

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

### Description

### Example

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

---

## ■ regress

Centred population regression for selected model variables

### Syntax

```
[B,COVRES,R2] = regress(M,LHS,RHS,...)
```

### Input arguments

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

### Output arguments

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



## Options

- 'output=' [ 'namedmat' | 'numeric' ] - Output matrices will be either namedmat objects or plain numeric arrays.

## Description

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

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

## Example

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

---

# ■ reporting

Run reporting equations

## Syntax

```
d = reporting(m,d,range,...)
```

## Input arguments

- `m` [ model ] - Model object with reporting equations.
- `d` [ struct ] - Input database that will be used to evaluate the reporting equations.
- `range` [ numeric ] - Date range on which the reporting equations will be evaluated.

## Output arguments

- `d` [ struct ] - Output database with reporting variables.

### Options

- 'dynamic=' [ true | false ] - If true, equations will be evaluated period by period allowing for own lags; if false, equations will be evaluated en bloc for all periods.
- 'merge=' [ true | false ] - Merge output database with input datase.

### Description

}

---

## ■ resample

Resample from the model implied distribution

### Syntax

```
OUTP = resample(M, INP, RANGE, NDRAW, ...)  
OUTP = resample(M, INP, RANGE, NDRAW, J, ...)
```

### Input arguments

- M [ model ] - Solved model object.
- INP [ struct | empty ] - Input data (if needed) for the distributions of initial condition and/or empirical shocks; if not bootstrap is invovled
- RANGE [ numeric ] - Resampling date range.
- NDRAW [ numeric ] - Number of draws.
- J [ struct | empty ] - Database user-supplied (time-varying) tunes on std devs, corr coeffs, and/or means of shocks.

### Output arguments

- OUTP [ struct | cell ] - Output database with resampled data.

## Options

- 'deviation=' [ true | false ] - Treat input and output data as deviations from balanced-growth path.
- 'dtrends=' [ 'auto' | true | false ] - Add deterministic trends to measurement variables.
- 'method=' [ 'bootstrap' | 'montecarlo' ] - Method of randomising shocks and initial condition.
- 'output=' [ 'auto' | 'dbase' | 'dpack' ] - Format of output data.
- 'progress=' [ true | false ] - Display progress bar in the command window.
- 'randomise=' [ true | false | numeric ] - Randomise initial condition.
- 'stateVector=' [ 'alpha' | 'x' ] - When resampling initial condition, use the transformed state vector, 'alpha', or the vector of original variables, 'x'; this option is meant to guarantee replicability of results.
- 'svdOnly=' [ true | false ] - Do not attempt Cholesky and only use SVD to factorize the covariance matrix when resampling initial condition; only applies when 'randomise=' true.
- 'wild=' [ true | false ] - Use wild bootstrap instead of Efron bootstrap; only applies when 'method=' 'bootstrap'.

## Description

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

## Example

---

## ■ set

Set modifiable model object properties

### Syntax

```
m = set(m,name,value)
m = set(m,name,value,name,value,...)
```

### Input arguments

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

### Output arguments

- `m [ model ]` - Model object with the requested property modified.

### Modifiable properties

#### *Labels*

- `'yLabels=' [ cellstr ]` - Labels added to measurement equations.
- `'xLabels=' [ cellstr ]` - Labels added to transition equations.
- `'dLabels=' [ cellstr ]` - Labels added to deterministic-trend equations.
- `'lLabels=' [ cellstr ]` - Labels added to dynamic links.

#### *Annotations*

- `'yAnnotations=' [ cellstr ]` - Annotations added to measurement equations.
- `'xAnnotations=' [ cellstr ]` - Annotations added to transition equations.
- `'eAnnotations=' [ cellstr ]` - Annotations added to shocks.
- `'pAnnotations=' [ cellstr ]` - Annotations added to parameters.

#### *Miscellaneous*

- `'nAlt=' [ numeric ]` - Number of alternative parameterisations.
- `'stdVec=' [ numeric ]` - Vector of std deviations.
- `'tOrigin=' [ numeric ]` - Base year for deterministic time trends in measurement variables.
- `'epsilon=' [ numeric ]` - Relative differentiation step when computing Taylor expansion.

## ■ shockplot

Short-cut for running and plotting plain shock simulation

### Syntax

```
[S,FF,AA] = shockplot(M,SHOCKNAME,RANGE,LIST,...)
```

### Input arguments

- M [ model ] - Model object that will be simulated.
- SHOCKNAME [ char ] - Name of the shock that will be simulated.
- RANGE [ numeric ] - Date range on which the shock will be simulated; the graphs will also include one pre-shock period.
- LIST [ cellstr ] - List of variables that will be reported; you can use the syntax of [dbase/dbplot](#) [P325](#).

### Output arguments

- S [ struct ] - Simulated database.
- FF [ numeric ] - Handles to figure windows created.
- AA [ numeric ] - Handles to axes objects created.

### Options affecting the simulation

- 'deviation=' [ [true](#) | false ] - See the option 'deviation=' in [model/simulate](#) [P118](#).
- 'dtrends=' [ [true](#) | false ] - See the option 'dtrends=' option in [model/simulate](#) [P118](#).
- 'shockSize=' [ '[std](#)' | numeric ] - Size of the shock that will be simulated; 'std' means that one std dev of the shock will be simulated.

### Options affecting the graphs

See help on [dbase/dbplot](#) [P325](#) for other options available.

## Description

## Example

---

## ■ simulate

Simulate model

## Syntax

```
S = simulate(M,D,RANGE,...)
[S,FLAG,ADDF,DISCREP] = simulate(M,D,RANGE,...)
```

## Input arguments

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

## Output arguments

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

## Output arguments in simulations with a non-linear plan

- FLAG [ cell ] - Cell array with exit flags for non-linearised simulations.
- ADDF [ cell ] - Cell array of tseries with final add-factors added to first-order approximate equations to make non-linear equations hold.
- DISCREP [ cell ] - Cell array of tseries with final discrepancies between LHS and RHS in equations earmarked for non-linear simulations by a double-equal sign.

## Options

- 'anticipate=' [ true | false ] - If true, real future shocks are anticipated, imaginary are unanticipated; vice versa if false.
- 'contributions=' [ true | false ] - Decompose the simulated paths into contributions of individual shocks.
- 'deviation=' [ true | false ] - Treat input and output data as deviations from balanced-growth path.
- 'dbExtend=' [ true | false | struct ] - Use the function dbextend to combine the simulated output data with the input database, or with another database, at the end.
- 'dTrends=' [ 'auto' | true | false ] - Add deterministic trends to measurement variables.
- 'ignoreShocks=' [ true | false ] - Read only initial conditions from input data, and ignore any shocks within the simulation range.
- 'plan=' [ plan ] - Specify a simulation plan to swap endogeneity and exogeneity of some variables and shocks temporarily, and/or to simulate some of the non-linear equations accurately.
- 'progress=' [ true | false ] - Display progress bar in the command window.

## Options for models with non-linearised equations

- 'addSstate=' [ true | false ] - Add steady state levels to simulated paths before evaluating non-linear equations; this option is used only if 'deviation=' true.
- 'display=' [ true | false | numeric ] - Display one-line report on each iteration; if 'display=' is a number n, display a report after every n iterations and a final report.
- 'error=' [ true | false ] - Throw an error whenever a non-linear simulation fails converge; if false, only an warning will display.
- 'lambda=' [ numeric | 1 ] - Step size (between 0 and 1) for add factors added to non-linearised equations in every iteration.
- 'reduceLambda=' [ numeric | 0.5 ] - Factor (between 0 and 1) by which lambda will be multiplied if the non-linear simulation gets on an divergence path.
- 'maxIter=' [ numeric | 100 ] - Maximum number of iterations.
- 'tolerance=' [ numeric | 1e-5 ] - Convergence tolerance.

## Description

The time series in the output database, S, are defined on the simulation range, RANGE, plus include any necessary initial conditions (for those variables that occur with lags in the model).

## Example

---

### ■ single

Convert solution matrices to single precision

## Syntax

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

## Input arguments

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

## Output arguments

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

## Description

---

### ■ solve

Calculate first-order accurate solution of the model

## Syntax

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



### Input arguments

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

### Output arguments

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

### Options

- `'expand=' [ numeric | 0 ]` - Number of periods ahead up to which the model solution will be expanded.
- `'progress=' [ true | false ]` - Display progress bar in the command window.
- `'refresh=' [ true | false ]` - Refresh dynamic links before computing the solution.
- `'select=' [ true | false ]` - Automatically detect which equations need to be re-differentiated based on parameter changes from the last time the model was solved.
- `'warning=' [ true | false ]` - Display warnings produced by this function.

### Description

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

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

## Example

---

### ■ srf

#### Shock response functions

#### Syntax

```
S = srf(M,NPER,...)
S = srf(M,RANGE,...)
```

#### Input arguments

- M [ model ] - Model object whose shock responses will be simulated.
- RANGE [ numeric ] - Simulation date range with the first date being the shock date.
- NPER [ numeric ] - Number of simulation periods.

#### Output arguments

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

#### Options

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

#### Description

#### Example

---

## ■ `sspace`

State-space matrices describing the model solution

### Syntax

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

### Input arguments

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

### Output arguments

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

### Options

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

### Description

The state-space representation has the following form:

$$[xf; \alpha] = T \cdot \alpha(-1) + K + R \cdot e$$

$$y = Z \cdot \alpha + D + H \cdot e$$

$$xb = U \cdot \alpha$$

$$\text{Cov}[e] = \Omega$$

where  $xb$  is an  $nb$ -by-1 vector of predetermined (backward-looking) transition variables and their auxiliary lags,  $xf$  is an  $nf$ -by-1 vector of non-predetermined (forward-looking) variables and their auxiliary leads,  $\alpha$  is a transformation of  $xb$ ,  $e$  is an  $ne$ -by-1 vector of shocks, and  $y$  is an  $ny$ -by-1 vector of measurement variables. Furthermore, we denote the total number of transition variables, and their auxiliary lags and leads,  $nx = nb + nf$ .

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

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

## ■ sstate

Compute steady state or balance-growth path of the model

### Syntax

```
m = sstate(m,...)
```

### Input arguments

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

### Output arguments

- `m [ model ]` - Model object with newly computed steady state assigned.

## Options

- 'warning=' [ true | false ] - Display IRIS warning produced by this function.

## Options for non-linear models

- 'blocks=' [ true | false ] - Re-arrange steady-state equations in recursive blocks before computing steady state.
- 'display=' [ 'iter' | 'final' | 'notify' | 'off' ] - Level of screen output, see Optim Tbx.
- 'endogenise=' [ cellstr | char | empty ] - List of parameters that will be endogenised when computing the steady state; the number of endogenised parameters must match the number of transition variables exogenised in the 'exogenise=' option.
- 'exogenise=' [ cellstr | char | empty ] - List of transition variables that will be exogenised when computing the steady state; the number of exogenised variables must match the number of parameters exogenised in the 'exogenise=' option.
- 'fix=' [ cellstr | empty ] - List of variables whose steady state will not be computed and kept fixed to the currently assigned values.
- 'fixAllBut=' [ cellstr | empty ] - Inverse list of variables whose steady state will not be computed and kept fixed to the currently assigned values.
- 'fixGrowth=' [ cellstr | empty ] - List of variables whose steady-state growth will not be computed and kept fixed to the currently assigned values.
- 'fixGrowthAllBut=' [ cellstr | empty ] - Inverse list of variables whose steady-state growth will not be computed and kept fixed to the currently assigned values.
- 'fixLevel=' [ cellstr | empty ] - List of variables whose steady-state levels will not be computed and kept fixed to the currently assigned values.
- 'fixLevelAllBut=' [ cellstr | empty ] - Inverse list of variables whose steady-state levels will not be computed and kept fixed to the currently assigned values.
- 'growth=' [ true | false ] - If true, both the steady-state levels and growth rates will be computed; if false, only the levels will be computed assuming that the model is either stationary or that the correct steady-state growth rates are already assigned in the model object.
- 'optimSet=' [ cell | empty ] - Name-value pairs with Optim Tbx settings; see help optimset for details on these settings.
- 'refresh=' [ true | false ] - Refresh dynamic links after steady state is computed.

- 'solver=' [ 'fsolve' | 'lsqnonlin' ] - Solver function used to solve for the steady state of non-linear models; it can be either of the two Optimization Tbx functions, or a user-supplied solver.
- 'sstate=' [ true | false | cell ] - If true or a cell array, the steady state is re-computed in each iteration; the cell array can be used to modify the default options with which the sstate function is called.

-Options for linear models

- 'refresh=' [ true | false ] - Refresh dynamic links before steady state is computed.
- 'solve=' [ true | false ] - Solve model before computing steady state.

## Description

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

## Example

---

## ■ sstatedb

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

### Syntax

```
D = sstatedb(M,RANGE)
D = sstatedb(M,RANGE,NCOL)
```

### Input arguments

- M [ model ] - Model object for which the sstate database will be created.
- RANGE [ numeric ] - Intended simulation range; the steady-state or balanced-growth-path database will be created on a range that also automatically includes all the necessary lags.

- `NCOL` [ numeric ] - Number of columns for each variable; the input argument `NCOL` can be only used with single-parameterisation models.

### Output arguments

- `D` [ struct ] - Database with a steady-state or balanced-growth path `tseries` object for each model variable, and a scalar or vector of the currently assigned values for each model parameter.

### Description

### Example

---

## ■ sstatefile

Create a steady-state file based on the model object's steady-state equations

### Syntax

```
sstatefile(m,filename,...)
```

### Input arguments

- `m` [ model ] - Model object.
- `file` [ char ] - Filename under which the steady-state file will be saved.

### Options

- `'endogenise='` [ cellstr | char | empty ] - List of parameters that will be endogenised when computing the steady state; the number of endogenised parameters must match the number of transtion variables exogenised in the `'exogenised='` option.
- `'endogenise='` [ cellstr | char | empty ] - List of transition variables that will be exogenised when computing the steady state; the number of exogenised variables must match the number of parameters exogenised in the `'exogenise='` option.
- `'growthNames='` [ char | 'd?' ] - Template for growth names used in evaluating lags and leads.

- 'time=' [ true | false ] - Keep or remove time subscripts (curly braces) in the steady-state file.

#### Description

#### Example

---

### ■ stdscale

Re-scale all std deviations by the same factor

#### Syntax

```
m = stdscale(m, factor)
```

#### Input arguments

- m [ model ] - Model object whose std deviations will be re-scaled.
- factor [ numeric ] - Factor by which all the model std deviations will be re-scaled.

#### Output arguments

- m [ model ] - Model object with all of its std deviations re-scaled.

#### Description

#### Example

---

### ■ subsasgn

Subscripted assignment for model and syeq objects



### Syntax for assigning parameterisations from other object

```
M(INDEX) = N
```

### Syntax for deleting specified parameterisations

```
M(INDEX) = []
```

### Syntax for assigning parameter values or steady-state values

```
M.NAME = X  
M(INDEX).NAME = X  
M.NAME(INDEX) = X
```

### Syntax for assigning std deviations or cross-correlations of shocks

```
M.STD_NAME = X  
M.CORR_NAME1__NAME2 = X
```

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

### Input arguments

- `M [ model | syeq ]` - Model or syeq object that will be assigned new parameterisations or new parameter values or new steady-state values.
- `N [ model | syeq ]` - Model or syeq object compatible with `M` whose parameterisations will be assigned (copied) into `M`.
- `INDEX [ numeric ]` - Index of parameterisations that will be assigned or deleted.
- `NAME, NAME1, NAME2 [ char ]` - Name of a variable, shock, or parameter.
- `X [ numeric ]` - A value (or a vector of values) that will be assigned to a parameter or variable named `NAME`.

### Output arguments

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

### Description

#### Example

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

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

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

---

## ■ subsref

Subscripted reference for model and syeq objects

### Syntax for retrieving object with subset of parameterisations

```
M(INDEX)
```

### Syntax for retrieving parameters or steady-state values

```
M.NAME
```

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

```
M.STD_NAME
```

```
M.CORR_NAME1__NAME2
```

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

### Input arguments

- `M [ model | syeq ]` - Model or syeq object.
- `INDEX [ numeric | logical ]` - Index of requested parameterisations.
- `NAME, NAME1, NAME2 [ char ]` - Name of a variable, shock, or parameter.

### Description

### Example

---

## ■ system

System matrices before model is solved

### Syntax

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

### Input arguments

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

### Output arguments

- `A [ numeric ]` - Matrix at the vector of expectations in the transition equation.
- `B [ numeric ]` - Matrix at current vector in the transition equations.
- `C [ numeric ]` - Constant vector in the transition equations.
- `D [ numeric ]` - Matrix at transition shocks in the transition equations.
- `F [ numeric ]` - Matrix at measurement variables in the measurement equations.
- `G [ numeric ]` - Matrix at predetermined transition variables in the measurement variables.
- `H [ numeric ]` - Constant vector in the measurement equations.
- `J [ numeric ]` - Matrix at measurement shocks in the measurement equations.

- `list [ cell ]` - Lists of measurement variables, transition variables including their auxiliary lags and leads, and shocks as they appear in the rows and columns of the system matrices.
- `nf [ numeric ]` - Number of non-predetermined (forward-looking) transition variables.

## Description

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

$$A E[xf;xb] + B [xf(-1);xb(-1)] + C + D e = 0$$

$$F y + G xb + H + J e = 0$$

where  $E$  is a conditional expectations operator,  $xf$  is a vector of non-predetermined (forward-looking) transition variables,  $xb$  is a vector of predetermined (backward-looking) transition variables,  $y$  is a vector of measurement variables, and  $e$  is a vector of transition and measurement shocks.

## Example

---

## ■ userdata

Get or set user data in an IRIS object

### Syntax for getting user data

```
X = userdata(OBJ)
```

### Syntax for assigning user data

```
OBJ = userdata(OBJ,X)
```

### Input arguments

- `OBJ [ model | tseries | VAR | SVAR | FAVAR | sstate ]` - One of the IRIS objects.
- `X [ ... ]` - Data that will be attached to the object.

### Output arguments

- `X [ ... ]` - User data that are currently attached to the object.

### Description

### Example

---

## ■ VAR

Population VAR for selected model variables

### Syntax

```
V = VAR(M,SELECT,RANGE,...)
```

### Input arguments

- `M [ model ]` - Solved model object.
- `SELECT [ cellstr | char ]` - List of variables selected for the VAR.
- `RANGE [ numeric ]` - Hypothetical range, including pre-sample initial condition, on which the VAR would be estimated.

### Output arguments

- `V [ VAR ]` - Asymptotic reduced-form VAR for selected model variables.

### Options

- `'order=' [ numeric | 1 ]` - Order of the VAR.
- `'constant=' [ true | false ]` - Include in the VAR a constant vector derived from the steady state of the selected variables.

## Description

## Example

---

### ■ vma

Vector moving average representation of the model

## Syntax

```
[PHI,LIST] = vma(M,P,...)
```

## Input arguments

- M [ model ] - Model object for which the VMA representation will be computed.
- P [ numeric ] - Order up to which the VMA will be evaluated.

## Output arguments

- PHI [ namedmat | numeric ] - VMA matrices.
- LIST [ cell ] - List of measurement and transition variables in the rows of the Phi matrix, and list of shocks in the columns of the Phi matrix.

## Option

- 'output=' [ 'namedmat' | numeric ] - Output matrix PHI will be either a namedmat object or a plain numeric array; if the option 'select=' is used, 'output=' is always 'namedmat'.
- 'select=' [ cellstr | Inf ] - Return the VMA matrices for selected variables and/or shocks only; Inf means all variables.

## Description

## Example

---

## ■ xsf

Power spectrum and spectral density of model variables

### Syntax

```
[S,D,LIST] = xsf(M,FREQ,...)
[S,D,LIST,FREQ] = xsf(M,NFREQ,...)
```

### Input arguments

- M [ model ] - Model object.
- FREQ [ numeric ] - Vector of frequencies at which the XSFs will be evaluated.
- NFREQ [ numeric ] - Total number of requested frequencies; the frequencies will be evenly spread between 0 and pi.

### Output arguments

- S [ namedmat | numeric ] - Power spectrum matrices.
- D [ namedmat | numeric ] - Spectral density matrices.
- LIST [ cellstr ] - List of variable in order of appearance in rows and columns of S and D.
- FREQ [ numeric ] - Vector of frequencies at which the XSFs has been evaluated.

### Options

- 'applyTo=' [ cellstr | char | Inf ] - List of variables to which the option 'filter=' will be applied; Inf means all variables.
- 'filter=' [ char | empty ] - Linear filter that is applied to variables specified by 'applyto'.
- 'nFreq=' [ numeric | 256 ] - Number of equally spaced frequencies over which the 'filter' is numerically integrated.
- 'output=' [ 'namedmat' | 'numeric' ] - Output matrices S and F will be either namedmat objects or plain numeric arrays; if the option 'select=' is used, 'output=' is always a namedmat object.
- 'progress=' [ true | false ] - Display progress bar on in the command window.

- 'select=' [ cellstr | char ] - Select only a subset of variables for which the XSF matrices will be returned.

## Description

## Example

---

## ■ zerodb

Create model-specific zero-deviation database

## Syntax

```
D = zerodb(M,RANGE)
D = zerodb(M,RANGE,NCOL)
```

## Input arguments

- M [ model ] - Model object for which the zero database will be created.
- RANGE [ numeric ] - Intended simulation range; the zero database will be created on a range that also automatically includes all the necessary lags.
- NCOL [ numeric ] - Number of columns for each variable; the input argument NCOL can be only used with single-parameterisation models.

## Output arguments

- D [ struct ] - Database with a tseries object filled with zeros for each linearised variable, a tseries object filled with ones for each log-linearised variables, and a scalar or vector of the currently assigned values for each model parameter.

## Description

## Example



## 6 Simulation plans

Simulation plans complement the use of the [model/simulate](#) P118 or [model/jforecast](#) P102 functions.

You need to use a simulation plan object to set up the following types of more complex simulations or forecasts (or a combination of these):

**simulations or forecasts with some of the model variables temporarily**

exogenised;

**simulations with some of the non-linear equations solved in an exact**

non-linear mode;

**forecasts conditioned upon some variables;**

The plan object is passed to the [model/simulate](#) P118 or [model/jforecast](#) P102 functions through the 'plan=' option.

Plan methods:

### Constructor

- [plan](#) P146 - Create new simulation plan.

### Getting information about simulation plans

- [detail](#) P139 - Display details of a simulation plan.
- [get](#) P141 - Query plan object properties.
- [nnzcond](#) P142 - Number of conditioning data points.
- [nnzendog](#) P143 - Number of endogenised data points.
- [nnzexog](#) P143 - Number of exogenised data points.
- [nnznonlin](#) P144 - Number of non-linearised data points.

### Setting up simulation plans

- [condition](#) P138 - Condition forecast upon the specified variables at the specified dates.
- [endogenise](#) P139 - Endogenise shocks or re-endogenise variables at the specified dates.
- [exogenise](#) P140 - Exogenise variables or re-exogenise shocks at the specified dates.
- [nonlinearise](#) P145 - Select equations for simulation in an exact non-linear mode.

### Referencing plan objects

- [subsref](#) P147 - Subscripted reference for plan objects.

### Getting on-line help on simulation plans

```
help plan  
help plan/function_name
```

---

## ■ condition

Condition forecast upon the specified variables at the specified dates

### Syntax

```
P = condition(M,LIST,DATES)
```

### Input arguments

- P [ plan ] - Simulation plan.
- LIST [ cellstr | char ] - List of variables upon which a forecast will be conditioned.
- DATES [ numeric ] - Dates at which the forecast will be conditioned upon the specified variables.

### Output arguments

- P [ plan ] - Simulation plan with new conditioning information included.

## Description

## Example

---

## ■ detail

Display details of a simulation plan

## Syntax

```
detail(P)
detail(P,D)
```

## Input arguments

- P [ plan ] - Simulation plan.
- D [ struct ] - Input database.

## Description

If you supply also the second input argument, an input database, both dates and values will be reported for exogenised and conditioning data points, and the values will be checked for the presence of NaNs (with a warning displayed should there be found any).

## Example

---

## ■ endogenise

Endogenise shocks or re-endogenise variables at the specified dates

## Syntax

```
P = endogenise(P,LIST,DATES)
P = endogenise(P,DATES,LIST)
```

## Simulation plans: exogenise

P = endogenise(P,LIST,DATES,FLAG)

P = endogenise(P,DATES,LIST,FLAG)

### Input arguments

- P [ plan ] - Simulation plan.
- LIST [ cellstr | char ] - List of shocks that will be endogenised, or list of variables that will be re-endogenise.
- DATES [ numeric ] - Dates at which the shocks or variables will be endogenised.
- FLAG [ 1 | li ] - Select the shock anticipation mode; if not specified 1 is used.

### Output arguments

- P [ plan ] - Simulation plan with new information on endogenised shocks included.

### Description

### Example

---

## ■ exogenise

Exogenise variables or re-exogenise shocks at the specified dates

### Syntax

P = exogenise(M,LIST,DATES)

P = exogenise(M,DATES,LIST)

P = exogenise(M,LIST,DATES,FLAG)

P = exogenise(M,DATES,LIST,FLAG)

### Input arguments

- P [ plan ] - Simulation plan.

## Simulation plans: get

- LIST [ cellstr | char ] - List of variables that will be exogenised, or list of shocks that will be re-exogenised.
- DATES [ numeric ] - Dates at which the variables will be exogenised.
- FLAG [ 1 | 1i ] - Select the shock anticipation mode (when re-exogenising shocks); if not specified 1 is used.

### Output arguments

- P [ plan ] - Simulation plan with new information on exogenised variables included.

### Description

### Example

---

## ■ get

Query plan object properties

### Syntax

```
ANS = get(P,QUERY)
[ANS,ANS,...] = get(P,QUERY,QUERY,...)
```

### Input arguments

- P [ plan ] - Simulation plan object.
- QUERY [ char ] - Name of the queried property.

### Output arguments

- ANS [ ... ] - Answer.

### Valid queries on plan objects

- 'endogenised=' — Returns [ struct ] a database with time series for each shock with 1 in each period where the variable is endogenised, and 0 in each period where the variable is not endogenised.
- 'exogenised=' — Returns [ struct ] a database with time series for each measurement and transition variable with 1 in each period where the variable is exogenised, and 0 in each period where the variable is not exogenised.
- 'onlyEndogenised=' — Returns [ struct ] the same database as 'endogenised' but including only those shocks that are endogenised at least in one period.
- 'onlyExogenised=' — Returns [ struct ] the same database as 'exogenised' but including only those measurement and transition variables that are endogenised at least in one period.
- 'range=' — Returns [ numeric ] the simulation plan range.

### Description

### Example

---

## ■ nnzcond

Number of conditioning data points

### Syntax

`N = nnzcond(P)`

### Input arguments

- `P` [ plan ] - Simulation plan.

### Output arguments

- `N` [ numeric ] - Number of conditioning data points; each variable at each date counts as one data point.

## Description

## Example

---

### ■ nnzendog

Number of endogenised data points

## Syntax

```
[N,NREAL,NIMAG] = nnzendog(P)
```

## Input arguments

- P [ plan ] - Simulation plan.

## Output arguments

- N [ numeric ] - Total number of endogenised data points; each shock at each time counts as one data point.
- NREAL [ numeric ] - Number of endogenised data points with anticipation mode 1.
- NIMAG [ numeric ] - Number of endogenised data points with anticipation mode 1i.

## Description

## Example

---

### ■ nnzexog

Number of exogenised data points

### Syntax

```
N = nnzexog(P)
```

### Input arguments

- P [ plan ] - Simulation plan.

### Output arguments

- N [ numeric ] - Number of exogenised data points; each variable at each date counts as one data point.

### Description

### Example

---

## ■ nnznonlin

Number of non-linearised data points

### Syntax

```
n = nnznonlin(p)
```

### Input arguments

- p [ plan ] - Simulation plan.

### Output arguments

- n [ numeric ] - Number of non-linearised equations.



## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ nonlinearise

Select equations for simulation in an exact non-linear mode

## Syntax

```
P = nonlinearise(P)
P = nonlinearise(P,RANGE)
P = nonlinearise(P,LIST,RANGE)
```

## Input arguments

- P [ plan ] - Simulation plan object.
- LIST [ cellstr ] - List of labels for equations that will be simulated in an exact non-linear mode; all selected equations must be marked by an `=#` P32 mark in the model file. If LIST is not specified, all equations marked in the model file will be simulated in a non-linear mode.
- RANGE [ numeric ] - Date range on which the equations will be simulated in an exact non-linear mode; currently, the range must start at the start of the plan range. If RANGE is not specified, the equations are non-linearised over the whole simulation range.

## Output arguments

- P [ plan ] - Simulation plan with information on exact non-linear simulation included.

## Description

## Example

---

## ■ plan

Create new simulation plan

### Syntax

```
p = plan(m,range)
```

### Input arguments

- `m` [ `model` ] - Model object that will be simulated subject to this simulation plan.
- `range` [ `numeric` ] - Simulation range; this range must exactly correspond to the range on which the model will be simulated.

### Output arguments

- `p` [ `plan` ] - New simulation plan.

### Description

You need to use a simulation plan object to set up the following types of more complex simulations or forecasts:

**simulations or forecasts with some of the model variables temporarily exogenised;**

**simulations with some of the non-linear equations solved exactly.**

**forecasts conditioned upon some variables;**

The plan object is passed to the [simulate](#) P118 or [jforecast](#) P102 functions through the option 'plan'.

### Example

---

## ■ subsref

Subscripted reference for plan objects

### Syntax

P = P(STARTDATE:ENDDATE)

### Input arguments

- P [ plan ] - Simulation plan.

### Output arguments

- P [ plan ] - Simulation plan reduced or expanded to the new range, STARTDATE:ENDDATE.
- STARTDATE [ numeric ] - New start date for the simulation plan.
- ENDDATE [ numeric ] - New end date for the simulation plan.

### Description

### Example

## 7 Posterior objects and functions

Posterior objects, `poster`, are used to evaluate the behaviour of the posterior distribution, and to draw model parameters from the posterior distribution.

Posterior objects are set up within the `model/estimate` [P76](#) function and returned as the second output argument - the set up and initialisation of the posterior object is fully automated in this case. Alternatively, you can set up a posterior object manually, by setting all its properties appropriately.

Poster methods:

### Constructor

- `poster` [P151](#) - Posterior objects and functions.

### Evaluating posterior density

- `arwm` [P148](#) - Adaptive random-walk Metropolis posterior simulator.
- `eval` [P150](#) - Evaluate posterior density at specified points.
- `testpar` [P153](#) - Test performance of for versus parfor loops in evaluating posterior simulator.

### Chain statistics

- `stats` [P152](#) - Evaluate selected statistics of an MCMC chain.

### Getting on-line help on model functions

```
help poster
help poster/function_name
```

---

## ■ arwm

Adaptive random-walk Metropolis posterior simulator

## Syntax

```
[THETA, LOGPOST, AR, SCALE, FINALCOV] = arwm(POS, NDRAW, ...)
```

## Input arguments

- POS [ poster ] - Initialised posterior simulator object.
- NDRAW [ numeric ] - Length of the chain not including burn-in.

## Output arguments

- THETA [ numeric ] - MCMC chain with individual parameters in rows.
- LOGPOST [ numeric ] - Vector of log posterior density (up to a constant) in each draw.
- AR [ numeric ] - Vector of cumulative acceptance ratios in each draw.
- SCALE [ numeric ] - Vector of proposal scale factors in each draw.
- FINALCOV [ numeric ] - Final proposal covariance matrix; the final covariance matrix of the random walk step is  $\text{SCALE}(\text{end})^2 \times \text{FINALCOV}$ .

## Options

- 'adaptProposalCov=' [ numeric | 0 ] - Speed of adaptation of the Cholesky factor of the proposal covariance matrix towards the target acceptance ratio, targetAR; zero means no adaptation.
- 'adaptScale=' [ numeric | 1 ] - Speed of adaptation of the scale factor to deviations of acceptance ratios from the target ratio, targetAR.
- 'burnin=' [ numeric | 0.10 ] - Number of burn-in draws entered either as a percentage of total draws (between 0 and 1) or directly as a number (integer greater than one). Burn-in draws will be added to the requested number of draws ndraw and discarded after the posterior simulation.
- 'estTime=' [ true | false ] - Display and update the estimated time to go in the command window.
- 'gamma=' [ numeric | 0.8 ] - The rate of decay at which the scale and/or the proposal covariance will be adapted with each new draw.

- 'initScale=' [ numeric | 1/3 ] - Initial scale factor by which the initial proposal covariance will be multiplied; the initial value will be adapted to achieve the target acceptance ratio.
  - 'progress=' [ true | false ] - Display progress bar in the command window.
- \*'targetAR=' [ numeric | 0.234 ] - Target acceptance ratio.

## Description

Use the [poster/stats](#) P152 function to process the raw chain produced by arwm.

## Example

---

## ■ eval

Evaluate posterior density at specified points

### Syntax

```
X = eval(POS)
X = eval(POS,P)
```

### Input arguments

- POS [ poster ] - Posterior object returned by the [model/estimate](#) P76 function.
- P [ numeric | cell ] - Parameter vector or a cell array of parameter vectors at which the posterior density will be evaluated; if P is not specified, the posterior density at the point of the estimated mode is returned.

### Output arguments

- X [ numeric ] - The value of posterior density evaluated at P; if P is a cell array the numeric array X is the same size.

## Description

## Example

---

## ■ poster

### Posterior objects and functions

Posterior objects, poster, are used to evaluate the behaviour of the posterior distribution, and to draw model parameters from the posterior distribution.

Posterior objects are set up within the [model/estimate](#) [P76](#) function and returned as the second output argument - the set up and initialisation of the posterior object is fully automated in this case. Alternatively, you can set up a posterior object manually, by setting all its properties appropriately.

Poster methods:

### Constructor

- [poster](#) [P151](#) - Posterior objects and functions.

### Evaluating posterior density

- [arwm](#) [P148](#) - Adaptive random-walk Metropolis posterior simulator.
- [eval](#) [P150](#) - Evaluate posterior density at specified points.
- [testpar](#) [P153](#) - Test performance of for versus parfor loops in evaluating posterior simulator.

### Chain statistics

- [stats](#) [P152](#) - Evaluate selected statistics of an MCMC chain.

### Getting on-line help on model functions

```
help poster
help poster/function_name
```

---

## ■ stats

Evaluate selected statistics of an MCMC chain

### Syntax

```
S = stats(POS,THETA,...)
S = stats(POS,THETA,LOGPOST,...)
```

### Input arguments

- POS [ poster ] - Posterior simulator, poster, object used to generate the THETA chain.
- THETA [ numeric ] - MCMC chain generated by the arwm function.
- LOGPOST [ numeric ] - Vector of log posterior densities generated by the arwm function; LOGPOST is not necessary if you do not request 'mdd', the marginal data density.

### Output arguments

- S [ struct ] - Struct with the statistics requested by the user.

### Options

- 'estTime=' [ true | false ] - Display and update the estimated time to go in the command window.
- 'histBins=' [ numeric | 10 ] - Number of histogram bins if 'hist' is in the 'output' option.
- 'hpdiCover=' [ numeric | 90 ] - Requested percent coverage of the highest probability density interval if 'hpdi' is in the 'output' option.
- 'mddGrid=' [ numeric | 0.1:0.1:0.9 ] - Points between 0 and 1 over which the marginal data density estimates will be averaged, see Geweke (1999).
- 'output=' [ char | cellstr ] - List of output statistics.
- 'progress=' [ true | false ] - Display progress bar in the command window.
- 'prctile=' [ numeric | [10,90] ] - Requested percentiles if 'prctile' is in the 'output' option.



## Description

List of available statistics:

—Overall statistics

- 'cov=' — Covariance matrix of the parameter estimates.
- 'mdd=' — Minus the log marginal data density.

—Parameter-specific statistics

- 'chain=' - The entire simulated chain of parameter values
- 'mean=' - Average.
- 'std=' - Std deviation.
- 'median=' - Median.
- 'mode=' - Mode based on the histogram function outputs.
- 'hpdi=' - Highest probability density interval with coverage given by the option 'hpdi'.
- 'hist=' - Histogram bins and counts, with the number of bins 'histbins'.
- 'ksdensity=' - The x- and y-axis points for kernel-smoothed posterior density.
- 'bounds=' - Lower and upper bounds set up by the user.
- 'prctile=' - Percentiles given by the option 'prctile'.

## Example

---

### ■ testpar

Test performance of for versus parfor loops in evaluating posterior simulator

## Syntax

```
[TFOR,TPARFOR] = testpar(POS,N,...)
```

### Input arguments

- POS [ poster ] - Initialised posterior simulator object that will be evaluated N time.
- N [ numeric ] - Number of times the posterior distribution will be evaluated for both for and parfor loops.

### Output arguments

- TFOR [ numeric ] - Time elapsed when running N times the posterior simulator with a for loop.
- TPARFOR [ numeric ] - Time elapsed when running N times the posterior simulator with a parfor loop.

### Options

- 'progress=' [ true | false ] - Display progress bar in the command window.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes & Troy Matheson.

## 8 Probability distribution package

This package gives quick access to basic univariate distributions. Its primary use is setting up priors in the [model/estimate](#) [P76](#) and [poster/arwm](#) [P148](#) functions.

The logdist package is called to create function handles that have several different modes of use. The primary use is to compute values that are proportional to the log of the respective density. In addition, the function handles also give you access to extra information (such as the the proper p.d.f., the name, mean, std dev, mode, and stuctural parameters of the distribution), and to a random number generator from the respective distribution.

Logdist methods:

### Getting function handles for univariate distributions

- [normal](#) [P159](#) - Create function proportional to log of normal distribution.
- [lognormal](#) [P158](#) - Create function proportional to log of log-normal distribution.
- [beta](#) [P156](#) - Create function proportional to log of beta distribution.
- [gamma](#) [P157](#) - Create function proportional to log of gamma distribution.
- [invgamma](#) [P157](#) - Create function proportional to log of inv-gamma distribution.
- [uniform](#) [P159](#) - Create function proportional to log of uniform distribution.

### Calling the logdist function handles

The function handles  $F$  created by the logdist package functions can be called the following ways:

- Get a value proportional to the log-density of the respective distribution at a particular point; this call is used within the [posterior simulator](#) [P148](#):  
 $y = F(x)$
- Get the density of the respective distribution at a particular point:  
 $y = F(x, 'pdf')$
- Get the characteristics of the distribution — mean, std deviation, mode, and information (the inverse of the second derivative of the log density):  
 $m = F([], 'mean')$   $s = F([], 'std')$   $o = F([], 'mode')$   $i = F([], 'info')$
- Get the underlying “structural” parameters of the respective distribution:  
 $a = F([], 'a')$   $b = F([], 'b')$

- Get the name of the distribution (the names correspond to the function names, i.e. can be either of 'normal', 'lognormal', 'beta', 'gamma', 'invgamma', 'uniform'):

```
name = F([], 'name')
```

- Draw a vector or matrix of random numbers from the distribution; drawing from beta, gamma, and inverse gamma requires the Statistics Toolbox:

```
a = F([], 'draw', 1, 1000);
```

```
size(a) ans = 1 10000
```

### Getting on-line help on logdist functions

```
help logdist
```

```
help logdist/function_name
```

---

## ■ beta

Create function proportional to log of beta distribution

### Syntax

```
F = logdist.beta(MEAN, STD)
```

### Input arguments

- MEAN [ numeric ] - Mean of the beta distribution.
- STD [ numeric ] - Std dev of the beta distribution.

### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the beta density.

### Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

## Example

---

### ■ gamma

Create function proportional to log of gamma distribution

#### Syntax

```
F = logdist.gamma(MEAN,STD)
```

#### Input arguments

- MEAN [ numeric ] - Mean of the gamma distribution.
- STD [ numeric ] - Std dev of the gamma distribution.

#### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the gamma density.

#### Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

## Example

---

### ■ invgamma

Create function proportional to log of inv-gamma distribution

#### Syntax

```
F = logdist.invgamma(MEAN,STD)
```

### Input arguments

- MEAN [ numeric ] - Mean of the inv-gamma distribution.
- STD [ numeric ] - Std dev of the inv-gamma distribution.

### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the inv-gamma density.

### Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

### Example

---

## ■ lognormal

Create function proportional to log of log-normal distribution

### Syntax

```
F = logdist.lognormal(MEAN,STD)
```

### Input arguments

- MEAN [ numeric ] - Mean of the log-normal distribution.
- STD [ numeric ] - Std dev of the log-normal distribution.

### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the log-normal density.

## Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

## Example

---

## ■ normal

Create function proportional to log of normal distribution

### Syntax

```
F = logdist.normal(MEAN,STD)
```

### Input arguments

- MEAN [ numeric ] - Mean of the normal distribution.
- STD [ numeric ] - Std dev of the normal distribution.

### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the normal density.

## Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

## Example

---

## ■ uniform

Create function proportional to log of uniform distribution

### Syntax

```
F = logdist.uniform(LOW,UPP)
```

### Input arguments

- LOW [ numeric ] - Lower bound of the uniform distribution.
- UPP [ numeric ] - Upper bound of the uniform distribution.

### Output arguments

- F [ function\_handle ] - Handle to a function returning a value that is proportional to the log of the uniform density.

### Description

See [help on the logdisk package](#) P155 for details on using the function handle F.

### Example



## 9 Steady-state file language

Steady-state (sstate) files are a complementary tool to solve and analyse the steady states of more complex models. They allow you to create steady-state files independent of the original model files, and write the steady-state equations in different ways, manipulate their structure or order, split the problem into subblocks, and combine numerical and symbolic solutions. Using then the [sstate objects](#) [P161], you can compile and run stand-alone steady-state m-file functions based on your steady-state files.

### Input parameters

- [!input](#) [P163] - List of input parameters or variables.
- [!growthnames](#) [P163] - Pattern for creating growth names.

### Equations and assignments

- [!equations](#) [P162] - Block of equations or assignments.
- [!growthnames2imag](#) [P163] - Pattern for creating growth names.
- [!solvefor](#) [P164] - List of variables for which the current equations block will be solved.
- [!symbolic](#) [P165] - Attempt to solve the current equations block symbolically using the Symbolic Maths Toolbox.

### Variables with steady state restricted to be positive

- [!log\\_variables](#) [P164] - Restrict the steady state of some of the variables to be positive.
- [!allbut](#) [P161] - Inverse list of variables with positive steady states.

### Getting on-line help on sstate file language

```
help sstatelang
help sstatelang/keyword
```

---

## ■ !allbut

Inverse list of variables with positive steady states

## Syntax

```
!log_variables
  !allbut
  LIST_OF_VARIABLES
```

## Description

See help on [!log\\_variables](#) P42.

---

## ■ !equations

Block of equations or assignments

### Syntax for a system of equations to be solved numerically

```
!equations
  EQUATION;
  EQUATION;
  EQUATION;
  ...

  !solvefor
  LIST_OF_VARIABLES
```

### Syntax for a system of equations to be solved symbolically

```
!equations
  EQUATION;
  EQUATION;
  EQUATION;
  ...

  !symbolic
  !solvefor
  LIST_OF_VARIABLES
```

### Syntax for block of assignments

```
!equations
  VARIABLE_NAME = EXPRESSION;
  VARIABLE_NAME = EXPRESSION;
  VARIABLE_NAME = EXPRESSION;
  ...
```

### Description

### Example

---

## ■ !growthnames

Pattern for creating growth names

### Syntax

```
!growthname := string_pattern;
!growthname := [function_pattern];
```

### Description

-Example

---

## ■ !input

List of input parameters or variables

### Syntax

```
!input
  parameter_or_variable_name, parameter_or_variable_name,
  parameter_or_variable_name, ...
```

## Description

## Example

---

### ■ !log\_variables

Restrict the steady state of some of the variables to be positive

## Syntax

```
!log_variables  
    list_of_variables
```

## Inverted syntax

```
!log_variables  
    !allbut  
    list_of_variables
```

## Description

## Example

---

### ■ !solvefor

List of variables for which the current equations block will be solved

## Syntax

```
!equations  
    EQUATION;
```

Steady-state file language: !symbolic

```
EQUATION;  
EQUATION;  
...  
  
!solvefor  
    LIST_OF_VARIABLES
```

## Description

## Example

---

## ■ !symbolic

Attempt to solve the current equations block symbolically using the Symbolic Maths Toolbox

## Syntax

```
!equations  
    EQUATION;  
    EQUATION;  
    EQUATION;  
    ...  
  
!symbolic  
!solvefor  
    LIST_OF_VARIABLES
```

## Description

## Example

## 10 Steady-state objects and functions

You can create a steady-state (sstate) object by loading a steady-state (sstate) file. The sstate object can be then saved as a stand-alone m-file function and repeatedly solved for different parameterisations.

Sstate methods:

### Constructor

- `sstate` P167 - Create new model object based on sstate file.

### Compiling stand-alone m-file functions

- `compile` P166 - Compile an m-file function based on a steady-state file.

### Running stand-alone sstate m-file functions

- `standalonemfile` P168 - Run a compiled stand-alone sstate m-file function.

### Getting on-line help on sstate functions

```
help sstate
help sstate/function_name
```

---

## ■ compile

Compile an m-file function based on a steady-state file

### Syntax

```
compile(S)
compile(S,FNAME,...)
```

### Input arguments

- `S [ sstate ]` - Sstate object built on a steady-state file.
- `FNAME [ char | empty ]` - Filename of the compiled m-file function; if not specified or empty the original steady-state filename will be used with an '.m' extension.

### Options

- `'excludeZero=' [ true | false ]` - Automatically detect and exclude zero solutions in blocks that result in multiple solutions.
- `'deleteSymbolicMfiles=' [ true | false ]` - Delete auxiliary m-files created to call the Symbolic Math toolbox.
- `'end=' [ numeric | char | Inf ]` - Compile the steady-state m-file function only up to this block.
- `'simplify=' [ numeric | Inf ]` - The minimum length for a symbolic expression to be simplified using the simplify function; Inf means no expressions will undergo simplification.
- `'start=' [ numeric | char | 1 ]` - Compile the steady-state m-file function starting from this block.
- `'symbolic=' [ true | false ]` - Call the Symbolic Math toolbox to solve blocks marked with a `!symbolic` keyword; otherwise, all blocks will be solved numerically regardless of their specification.

### Description

### Example

---

## ■ sstate

Create new model object based on sstate file

### Syntax

```
S = sstate(FNAME,...)
```

### Input arguments

- `FNAME [ char ]` - Name of the steady-state file that will be loaded and converted to a new `sstate` object.

### Output arguments

- `S [ sstate ]` - New `sstate` object based on the input steady-state file.

### Options

- `'assign=' [ struct | empty ]` - Database that will be used by the preparer to evaluate conditions and expressions in the `!if` and `!switch` structures.

### Description

### Example

---

## ■ `standalonemfile`

Run a compiled stand-alone `sstate` m-file function

After you [`sstate/compile`](#) `P166` the stand-alone `sstate` m-file function, type `help FNAME` to get help on running the file, where `FNAME` is the name of the file created.



## 11 Matrices with named rows and columns

Matrices with named rows and columns are returned by several IRIS functions, such as `model/acf` [P66], `model/xsf` [P135], or `model/fmse` [P89], to facilitate easy selection of submatrices by referring to variable names in rows and columns.

Namedmat methods:

### Constructor

- `namedmat` [P169] - Create a new matrix with named rows and columns.

### Manipulating named matrices

- `select` [P170] - Select submatrix by referring to row names and column names.
- `transpose` [P171] - Transpose each page of matrix with names rows and columns.

All operators and functions available for standard Matlab matrices and arrays (i.e. double objects) are also available for `namedmat` objects.

## ■ `namedmat`

Create a new matrix with named rows and columns

### Syntax

```
X = namedmat(X,ROWNAMES,COLNAMES)
```

### Input arguments

- `X` [ numeric ] - Matrix or multidimensional array.
- `ROWNAMES` [ cellstr ] - Names for individual rows of `X`.
- `COLNAMES` [ cellstr ] - Names for individual columns of `X`.

### Output arguments

- X [ namedmat ] - Matrix with named rows and columns.

### Description

Namedmat objects are used by some of the IRIS functions to preserve the names of variables that relate to individual rows and columns, such as in

- acf, the autocovariance and autocorrelation functions,
- xsf, the power spectrum and spectral density functions,
- fmse, the forecast mean square error functions,
- etc.

You can use the function [select](#) P170 to extract submatrices by referring to a selection of names.

Namedmat matrices derives from the built-in double class of objects, and hence you can use any operators and functions on them that are available for double objects.

### Example

---

## ■ select

Select submatrix by referring to row names and column names

### Syntax

```
[X,INDEX] = select(X,ROWSELECT,COLSELECT)
```

```
[X,INDEX] = select(X,SELECT)
```

### Input arguments

- X [ namedmat ] - Matrix or array with named rows and columns.
- ROWSELECT [ char | cellstr ] - Selection of row names.

- COLSELECT [ char | cellstr ] - Selection of column names.
- SELECT [ char | cellstr ] - Selection of names that will be applied to both rows and columns.

#### Output arguments

- X [ namedmat ] - Submatrix with named rows and columns.

#### Description

#### Example

---

## ■ transpose

Transpose each page of matrix with names rows and columns

#### Syntax

```
X = transpose(X)
X = X.'
```

#### Input arguments

- X [ namedmat ] - Input matrix or array with named rows and columns.

#### Output arguments

- X [ namedmat ] - Transpose of the input matrix; if it is more than 2-dimensional, each page of the matrix is transposed.

#### Description

#### Example

Part III —  
Multivariate time series analysis

## 12 Vector autoregressions: VAR objects and functions

VAR methods:

### Constructor

- [VAR](#) P197 - Create new reduced-form VAR object.

### Getting information about VAR objects

- [comment](#) P176 - Get or set user comments in an IRIS object.
- [companion](#) P177 - Matrices of first-order companion VAR.
- [eig](#) P179 - Eigenvalues of a VAR process.
- [get](#) P184 - Query VAR object properties.
- [isexplosive](#) P188 - True if any eigenvalue is outside unit circle.
- [isstationary](#) P189 - True if all eigenvalues are within unit circle.
- [length](#) P189 - Number of alternative parameterisations in VAR object.
- [mean](#) P191 - Mean of VAR process.
- [sspace](#) P194 - Quasi-triangular state-space representation of VAR.
- [userdata](#) P196 - Get or set user data in an IRIS object.

### Referencing VAR objects

- [subsasgn](#) P195 - Subscripted assignment for VAR objects.
- [subsref](#) P196 - Subscripted reference for VAR objects.

### Simulation and forecasting

- [ferf](#) P182 - Forecast error response function.
- [forecast](#) P183 - Unconditional or conditional forecasts.
- [instrument](#) P185 - Define conditioning instruments for VAR model.
- [resample](#) P192 - Resample from a VAR object.
- [simulate](#) P193 - Simulate VAR model.

## Manipulating VARs

- `alter` P175 - Expand or reduce number of alternative parameterisations.
- `backward` P176 - Backward VAR process.
- `demean` P178 - Remove constant from VAR object and demean associated data.
- `horzcat` P185 - Combine two compatible VAR objects in one object with multiple parameterisations.
- `integrate` P187 - Integrate VAR process and data associated with it.

## Stochastic properties

- `acf` P174 - Autocovariance and autocorrelation functions for VAR variables.
- `fmse` P182 - Forecast mean square error matrices.
- `vma` P198 - Matrices describing the VMA representation of a VAR process.
- `xsf` P199 - Power spectrum and spectral density functions for VAR variables.

## Estimation, identification, and statistical tests

- `estimate` P179 - Estimate a reduced-form VAR or BVAR.
- `lrtest` P190 - Likelihood ratio test for VAR models.
- `portest` P191 - Portmanteau test for autocorrelation in VAR residuals.

## Getting on-line help on VAR functions

```
help VAR
help VAR/function_name
```

---

## ■ acf

Autocovariance and autocorrelation functions for VAR variables

## Syntax

```
[C,R] = acf(V,...)
```

## Input arguments

- `V` [ VAR ] - VAR object for which the ACF will be computed.

## Output arguments

- `C` [ numeric ] - Auto/cross-covariance matrices.
- `R` [ numeric ] - Auto/cross-correlation matrices.

## Options

- `'applyTo='` [ logical | Inf ] - Logical index of variables to which the `'filter='` will be applied; the default `Inf` means all variables.
- `'filter='` [ char | empty ] - Linear filter that is applied to variables specified by `'applyto'`.
- `'nfreq='` [ numeric | 256 ] - Number of equally spaced frequencies over which the `'filter'` is numerically integrated.
- `'order='` [ numeric | 0 ] - Order up to which ACF will be computed.
- `'progress='` [ true | false ] - Display progress bar in the command window.

## Description

## Example

---

## ■ alter

Expand or reduce number of alternative parameterisations

## Syntax

```
V = alter(V,N)
```

### Input arguments

- `v [ VAR ]` - VAR object in which the number of parameterisations will be changed.
- `N [ numeric ]` - New number of parameterisations.

### Output arguments

- `v [ VAR ]` - VAR object with the new number of parameterisations.

### Description

### Example

---

## ■ backward

Backward VAR process

### Syntax

```
v = backward(v)
```

### Input arguments

- `v [ VAR ]` - VAR object.

### Output arguments

- `v [ VAR ]` - VAR object with the VAR process reversed in time.
- 

## ■ comment

Get or set user comments in an IRIS object



### Syntax for getting user comments

```
C = comment(OBJ)
```

### Syntax for assigning user comments

```
OBJ = comment(OBJ,C)
```

### Input arguments

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

### Output arguments

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

### Description

### Example

---

## ■ companion

Matrices of first-order companion VAR

### Syntax

```
[A,B,K] = companion(v)
```

### Input arguments

- v [ VAR ] - VAR object for which the companion matrices will be returned.

### Output arguments

- A [ numeric ] - First-order companion transition matrix.
- B [ numeric ] - First-order companion coefficient matrix at reduced-form residuals.
- K [ numeric ] - First-order compnaion constant vector.

### Description

### Example

```
}
```

---

## ■ demean

Remove constant from VAR object and demean associated data

### Syntax

```
V = demean(V,...)
[V,D] = demean(V,DATA,...)
```

### Input arguments

- V [ VAR ] - VAR object in which the constant vector will be reset to zero.
- D [ struct | tseries ] - Datase or tseries associated with the VAR object from which the unconditional mean will be removed.

### Output arguments

- V [ VAR ] - VAR object with the constant vector reset to zero.
- D [ struct | tseries ] - Demeaned data.

## Description

## Example

---

### ■ eig

Eigenvalues of a VAR process

## Syntax

```
e = eig(v)
```

## Input arguments

- `v [ VAR ]` - VAR object whose eigenvalues will be returned.

## Output arguments

- `e [ numeric ]` - VAR eigenvalues.

## Description

This function is equivalent to calling

```
e = get(v, 'eig')
```

## Example

```
}
```

---

### ■ estimate

Estimate a reduced-form VAR or BVAR

### Syntax with database input

```
[V,DATA,FITTED] = estimate(W,D,RANGE,...)
[V,DATA,FITTED] = estimate(W,D,LIST,RANGE,...)
```

### Syntax with tseries input

```
[V,DATA,FITTED] = estimate(W,X,RANGE,...)
```

### Input arguments

- V [ VAR ] - Empty VAR object.
- D [ struct ] - Input database.
- LIST [ cellstr ] - List of series from the database on which the VAR will be estimated; LIST can be omitted if variable names have been already specified at the time of creating the VAR object.
- X [ tseries ] - Tseries objects with input data.
- RANGE [ numeric ] - Estimation range, including P pre-sample observations where P is the order of the VAR.

### Output arguments

- W [ VAR ] - Estimated reduced-form VAR object.
- DATA [ struct | tseries ] - Output data that include the endogenous variables and the estimated residuals; the output data is a struct (database) when estimate is called with a database input, and the output data is a tseries when estimate is called with a tseries input.
- FITTED [ numeric ] - Periods for which fitted values can be calculated.

### Options

- 'A=' [ numeric | empty ] - Restrictions on the individual values in the transition matrix, A.
- 'BVAR=' [ numeric ] - Prior dummy observations for estimating a BVAR; construct the dummy observations using the one of the BVAR functions.
- 'C=' [ numeric | empty ] - Restrictions on the individual values in the constant vector, C.

- 'diff=' [ true | false ] - Difference the series before estimating the VAR; integrate the series back afterwards.
- 'G=' [ numeric | empty ] - Restrictions on the individual values in the matrix at the co-integrating vector, G.
- 'cointeg=' [ numeric | empty ] - Co-integrating vectors (in rows) that will be imposed on the estimated VAR.
- 'constraints=' [ char ] - General linear constraints on the VAR parameters.
- 'constant=' [ true | false ] - Include a constant vector in the VAR.
- 'covParameters=' [ true | false ] - Calculate the covariance matrix of estimated parameters.
- 'eqtnByEqtn=' [ true | false ] - Estimate the VAR equation by equation.
- 'maxIter=' [ numeric | 1 ] - Maximum number of iterations when generalised least squares algorithm is involved.
- 'mean=' [ numeric | empty ] - Assume a particular mean of the VAR process.
- 'order=' [ numeric | 1 ] - Order of the VAR.
- 'progress=' [ true | false ] - Display progress bar in the command window.
- 'schur=' [ true | false ] - Calculate triangular (Schur) representation of the estimated VAR.
- 'stdize=' [ true | false ] - Adjust the prior dummy observations by the std dev of the observations.
- 'tolerance=' [ numeric | 1e-5 ] - Convergence tolerance when generalised least squares algorithm is involved.
- 'yNames=' [ cellstr | function\_handle | @(n) sprintf('y%g',n) ] - Use these names for the VAR variables.
- 'eNames=' [ cellstr | function\_handle | @(yname,n) ['res\_',yname] ] - Use these names for the VAR residuals.
- 'warning=' [ true | false ] - Display warnings produced by this function.

## Description

If the VAR is estimated with a database input, D, the variable names, LIST, will be stored within the VAR object, and used whenever you call a VAR or SVAR function that returns output data, such as [VAR/simulate](#) [P193](#) or [SVAR/srf](#) [P207](#).

If the VAR is estimated with a tseries input, X, the VAR functions will return tseries as output data.

## Example

---

### ■ ferf

Forecast error response function

#### Syntax

```
[Phi,Psi,s,c] = ferf(v,nper)
[Phi,Psi,s,c] = ferf(v,range)
```

#### Output arguments

- Phi [ numeric ] - Response function matrices.
- Psi [ numeric ] - Cumulative response function matrices.
- s [ tseries ] - Response function time series.
- c [ tseries ] - Cumulative response function time series.

#### Input arguments

- v [ VAR ] - VAR object for which the forecast error response function will be computed.
- nper [ numeric ] - Number of periods.
- range [ numeric ] - Date range.

#### Description

#### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

### ■ fmse

Forecast mean square error matrices

### Syntax

```
[M,x] = fmse(this,nper)
[M,x] = fmse(this,range)
```

### Input arguments

- `v` [ VAR ] - VAR object for which the forecast MSE matrices will be computed.
- `nper` [ numeric ] - Number of periods.
- `range` [ numeric ] - Date range.

### Output arguments

- `M` [ numeric ] - Forecast MSE matrices.
- `x` [ dbase | tseries ] - Database or tseries with the std deviations of individual variables, i.e. the square roots of the diagonal elements of `M`.

### Options

- `'output='` [ 'dbase' | 'tseries' ] - Format of output data.

### Description

### Example

```
}
```

---

## ■ forecast

Unconditional or conditional forecasts

### Syntax

```
[OUTP,INST] = forecast(V,INP,RANGE,J,...)
```

### Input arguments

- `v` [ VAR ] - VAR object.
- `INP` [ struct | tseries ] - Input data that will be used to set up initial condition for the forecast.
- `RANGE` [ numeric ] - Forecast range.
- `J` [ struct | tseries ] - Conditioning database with structural tunes on the mean of residuals, reduced-form tunes on the endogenous variables, and tunes on the conditioning instruments.

### Output arguments

- `OUTP` [ struct | tseries ] - Output database or output tseries object.
- `INST` [ struct | empty ] - Output database with paths for conditioning instruments.

### Description

### Example

---

## ■ `get`

Query VAR object properties

### Syntax

```
value = get(v,query)
[value,value,...] = get(v,query,query,...)
```

### Input arguments

- `v` [ VAR ] - VAR object.
- `query` [ char ] - Name of the queried property.

### Output arguments

- `value` [ ... ] - Value of the queried property.



## Valid queries on VAR objects

### Description

### Example

---

## ■ horzcat

Combine two compatible VAR objects in one object with multiple parameterisations

### Syntax

$v = [v1, v2, \dots]$

### Input arguments

- $v1, v2$  [ VAR ] - Compatible VAR objects that will be combined.

### Output arguments

- $v$  [ VAR ] - Output VAR object that combines the input VAR objects as multiple parameterisations.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ instrument

Define conditioning instruments for VAR model

### Syntax to add forecast instruments

```
V = instrument(V,DEF)
```

### Syntax to remove all forecast instruments

```
V = instrument(V)
```

### Input arguments

- `V [ VAR ]` - VAR object to which forecast instruments will be added.
- `DEF [ char | cellstr ]` - Definition string for the conditioning instruments.

### Output arguments

- `V [ VAR ]` - VAR object with forecast instruments added or removed.

### Description

Conditioning instruments allow you to compute forecasts conditional upon a linear combination of endogenous variables.

The definition strings must have the following form:

```
'name := expression'
```

where `name` is the name of the new conditioning instrument, and `expression` is an expression referring to existing VAR variable names and/or their lags.

The conditioning instruments must be a linear combination (possibly with a constant) of the existing endogenous variables. The names of the conditioning instruments must be obviously unique (i.e. distinct from the names of the existing endogenous variables, and from other instruments).

### Example

In the following example, we assume that the VAR object `v` has at least three endogenous variables named `x`, `y`, and `z`.

```
V = instrument(V,'i1 := x - x{-1}','i2: = (x + y + z)/3');
```

Note that the above line of code is equivalent to

```
V = instrument(V,'i1 := x - x{-1}');
V = instrument(V,'i2: = (x + y + z)/3');
```

The command defines two conditioning instruments named i1 and i2. The first instrument is the first difference of the variable x. The second instrument is the average of the three endogenous variables.

To impose conditions (tunes) on a forecast using these instruments, you run [VAR/forecast](#) P183 with the fourth input argument containing a time series for i1, i2, or both.

```
j = struct();
j.i1 = tseries(startdate:startdate+3,0);
j.i2 = tseries(startdate:startdate+3,[1;1.5;2]);

f = forecast(v,d,startdate:startdate+12,j);
```

## ■ integrate

Integrate VAR process and data associated with it

### Syntax

```
V = integrate(V,...)
[V,D] = integrate(V,D,...)
```

### Input arguments

- V [ VAR ] - VAR object whose variables will be integrated.
- D [ tseries ] - Database or tseries associated with the VAR object.

### Output arguments

- `V [ VAR ]` - VAR object with (some of) its variables integrated.
- `D [ struct | tseries ]` - Integrated data associated with the VAR object.

### Options

- `'applyTo=' [ logical | numeric | Inf ]` - Index of variables to integrate; `Inf` means all variables will be integrated.

### Description

### Example

---

## ■ isexplosive

True if any eigenvalue is outside unit circle

### Syntax

```
FLAG = isexplosive(V)
```

### Input arguments

- `V [ VAR ]` - VAR object.

### Output arguments

- `FLAG [ true | false ]` - True if any eigenvalue is outside unit circle.

### Options

- `'tolerance=' [ numeric | getrealsmall() ]` - Tolerance for the eigenvalue test.

## Description

## Example

---

### ■ **isstationary**

True if all eigenvalues are within unit circle

## Syntax

```
flag = isstationary(v)
```

## Input arguments

- `v [ VAR ]` - VAR object.

## Output arguments

- `flag [ true | false ]` - True if all eigenvalues are within unit circle.

## Options

- `'tolerance=' [ numeric | getrealsmall() ]` - Tolerance for the eigenvalue test.
- 

### ■ **length**

Number of alternative parameterisations in VAR object

## Syntax

```
n = length(w)
```

### Input arguments

- `v [ VAR ]` - VAR object.

### Output arguments

- `n [ numeric ]` - Number of alternative parameterisations.

### Description

### Example

---

## ■ lrtest

Likelihood ratio test for VAR models

### Syntax

```
[stat,crit] = lrtest(V1,V2,level)
```

### Input arguments

- `V1 [ VAR ]` - Unrestricted VAR model.
- `V2 [ VAR ]` - Restricted VAR model.
- `level [ numeric ]` - Significance level; if not specified, 0.05 is used.

### Output arguments

- `stat [ numeric ]` - LR test stastic.
- `crit [ numeric ]` - LR test critical value based on chi-square distribution.

### Description

### Example

---

## ■ mean

Mean of VAR process

### Syntax

```
x = mean(v)
```

### Input arguments

- v [ VAR ] - VAR object.

### Output arguments

- x [ numeric ] - Mean vector.

### Description

### Example

---

## ■ portest

Portmanteau test for autocorrelation in VAR residuals

### Syntax

```
[STAT,CRIT] = portest(V,DATA,H)
```

### Input arguments

- V [ VAR | swar ] - Estimated VAR from which the tested residuals were obtained.
- DATA [ tseries ] - VAR residuals, or VAR output data including residuals, to be tested for autocorrelation.
- H [ numeric ] - Test horizon; must be greater than the order of the tested VAR.

### Output arguments

- STAT [ numeric ] - Portmanteau test statistic.
- CRIT [ numeric ] - Portmanteau test critical value based on chi-square distribution.

### Options

- 'level=' [ numeric | 0.05 ] - Requested significance level for computing the criterion CRIT.

### Description

### Example

---

## ■ resample

Resample from a VAR object

### Syntax

```
d = resample(w,d,range,ndraw,...)
d = resample(w,[],range,ndraw,...)
```

### Input arguments

- w [ VAR ] - VAR object to resample from.
- input [ struct | tseries ] - Input database or tseries used in bootstrap; not needed when 'method=' 'montecarlo'.
- range [ numeric ] - Range for which data will be returned.

### Output arguments

- d [ struct | tseries ] - Resampled output database or tseries.



### Options

- 'method=' [ 'bootstrap' | 'montecarlo' | function\_handle ] - Bootstrap from estimated residuals, resample from normal distribution, or use user-supplied sampler.
- 'progress=' [ true | false ] - Display progress bar in the command window.
- 'randomise=' [ true | false ] - Randomise or fix pre-sample initial condition.
- 'wild=' [ true | false ] - Use wild bootstrap instead of standard Efron bootstrap when 'method=' 'bootstrap'.

### Description

### Example

---

## ■ simulate

Simulate VAR model

### Syntax

```
OUTP = simulate(V,INP,RANGE,...)
```

### Input arguments

- V [ VAR ] - VAR object that will be simulated.
- INP [ tseries | struct ] - Input data from which the initial conditions and residuals will be taken.
- RANGE [ numeric ] - Simulation range.

### Output arguments

- OUTP [ tseries ] - Simulated output data.

### Options

- `'contributions='` [ `true` | `false` ] - Decompose the simulated paths into contributions of individual residuals.
- `'deviation='` [ `true` | `false` ] - Treat input and output data as deviations from unconditional mean.
- `'output='` [ `'auto'` | `'dbase'` | `'tseries'` ] - Format of output data.

### Description

### Example

---

## ■ `sspace`

Quasi-triangular state-space representation of VAR

### Syntax

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

### Input arguments

- `w` [ VAR ] - VAR object.

### Output arguments

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

- `Omega [ numeric ]` - Covariance matrix of shocks.

## Description

## Example

-IRIS Toolbox. -Copyright 2007–2012 Jaromir Benes.

---

## ■ subsasgn

Subscripted assignment for VAR objects

Syntax to assign parameterisations from other VAR object

```
v(index) = w
```

Syntax to delete specified parameterisations

```
v(index) = []
```

## Input arguments

- `v [ VAR ]` - VAR object.
- `index [ numeric ]` - Index of parameterisations that will be assigned or deleted.
- `w [ VAR ]` - VAR object compatible with `m` whose parameterisations will be assigned (copied) into `v`.

## Output arguments

- `v [ model ]` - VAR object with newly assigned or deleted parameterisations,

## Description

### Example

Expand the number of parameterisations in a VAR object that has initially just one parameterisation:

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

The parameterisation is simply copied ten times within the VAR object.

---

## ■ subsref

Subscripted reference for VAR objects

Syntax to retrieve VAR object with subset of parameterisations

```
v(index)
```

### Input arguments

- `v [ VAR ]` - VAR object.
- `index [ numeric | logical ]` - Index of requested parameterisations.

## Description

### Example

---

## ■ userdata

Get or set user data in an IRIS object

### Syntax for getting user data

```
X = userdata(OBJ)
```

### Syntax for assigning user data

```
OBJ = userdata(OBJ,X)
```

### Input arguments

- OBJ [ model | tseries | VAR | SVAR | FAVAR | sstate ] - One of the IRIS objects.
- X [ ... ] - Data that will be attached to the object.

### Output arguments

- X [ ... ] - User data that are currently attached to the object.

### Description

### Example

---

## ■ VAR

Create new reduced-form VAR object

### Syntax

```
V = VAR()  
V = VAR(YNAMES)  
V = VAR(YNAMES,ENAMES)
```

### Output arguments

- `V [ VAR ]` - New empty VAR object.
- `YNAMES [ cellstr | char | function_handle ]` - Names of VAR variables.
- `ENAMES [ cellstr | char | function_handle ]` - Names of VAR residuals.

### Description

This function creates a new empty VAR object. It is usually followed by the [estimate](#) P179 function to estimate the VAR parameters on data.

### Example

To estimate a VAR, you first need to create an empty VAR object, and then run the [VAR/estimate](#) P179 function on it, e.g.

```
V = VAR();  
[V,D] = estimate(V,D,list,range);
```

or simply

```
[V,D] = estimate(V,D,list,range);
```

---

## ■ vma

Matrices describing the VMA representation of a VAR process

### Syntax

```
PHI = vma(V,N)
```

### Input arguments

- `V` [ VAR ] - VAR object for which the VMA matrices will be computed.
- `N` [ numeric ] - Order up to which the VMA matrices will be computed.

### Output arguments

- `PHI` [ numeric ] - VMA matrices.

### Description

### Example

---

## ■ xsf

Power spectrum and spectral density functions for VAR variables

### Syntax

`[S,D] = xsf(V,FREQ,...)`

### Input arguments

- `V` [ VAR ] - VAR object.
- `FREQ` [ numeric ] - Vector of Frequencies at which the XSFs will be evaluated.

### Output arguments

- `S` [ numeric ] - Power spectrum matrices.
- `D` [ numeric ] - Spectral density matrices.

## Options

- 'applyTo=' [ cellstr | char | ':' ] - List of variables to which the 'filter=' will be applied; ':' means all variables.
- 'filter=' [ char | empty ] - Linear filter that is applied to variables specified by 'applyto'.
- 'nFreq=' [ numeric | 256 ] - Number of equally spaced frequencies over which the 'filter' is numerically integrated.
- 'progress=' [ true | false ] - Display progress bar in the command window.

## Description

The output matrices, S and D, are N-by-N-by-K, where N is the number of VAR variables and K is the number of frequencies (i.e. the length of the vector freq).

The k-th page is the S matrix, i.e.  $S(:, :, k)$ , is the cross-spectrum matrix for the VAR variables at the k-th frequency. Similarly, the k-th page in D, i.e.  $D(:, :, k)$ , is the cross-density matrix.

## Example



## 13 Structural vector autoregressions: SVAR objects and functions

SVAR methods:

### Constructor

- [SVAR](#) P208 - Convert reduced-form VAR to structural VAR.

SVAR objects can call any of the [VAR](#) P173 functions. In addition, the following functions are available for SVAR objects.

### Getting information about SVAR objects

- [fprintf](#) P203 - Format SVAR as a model code and write to text file.
- [get](#) P203 - Query SVAR object properties.
- [sprintf](#) P206 - Format SVAR as a model code and write to text string.

### Simulation

- [srf](#) P207 - Shock (impulse) response function.

### Stochastic properties

- [fevd](#) P202 - Forecast error variance decomposition for SVAR variables.

### Manipulating SVAR objects

- [sort](#) P204 - Sort SVAR parameterisations by squared distance of shock responses to median.

See help on [VAR](#) P173 objects for other functions available.

### Getting on-line help on SVAR functions

```
help SVAR
help SVAR/function_name
```

## Getting on-line help on SVAR functions that are inherited from VARs

```
help VAR
help VAR/function_name
```

---

## ■ fevd

Forecast error variance decomposition for SVAR variables

### Syntax

```
[X,Y,x,y] = fevd(V,NPER)
[X,Y,x,y] = fevd(V,RANGE)
```

### Input arguments

- V [ VAR ] - Structural VAR model.
- NPER [ numeric ] - Number of periods.
- RANGE [ numeric ] - Date range.

### Output arguments

- X [ numeric ] - Forecast error variance decomposition into absolute contributions of residuals; absolute contributions sum up to the total variance.
- Y [ numeric ] - Forecast error variance decomposition into relative contributions of residuals; relative contributions sum up to 1.
- x [ tseries ] - X converted to a tseries object.
- y [ tseries ] - Y converted to a tseries object.

### Description

### Example

---

## ■ fprintf

Format SVAR as a model code and write to text file

### Syntax

```
[c,d] = fprintf(s,fname,...)
```

### Input arguments

- `s` [ SVAR ] - SVAR object that will be written as a model code.
- `fname` [ char | cellstr ] - Filename, or filename format string, under which the model code will be saved.
- Output arguments
- `c` [ cellstr ] - Text string with the model code for each parameterisation.
- `d` [ cell ] - Parameter databases for each parameterisation; if 'hardParameters=' true, the database will be empty.

### Options

See help on [sprintf](#) P206 for options available.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ get

Query SVAR object properties

### Syntax

```
value = get(v,query)
[value,value,...] = get(v,query,query,...)
```

### Input arguments

- `v [ SVAR ]` - SVAR object.
- `query [ char ]` - Name of the queried property.

### Output arguments

- `value [ ... ]` - Value of the queried property.

All properties accessible through the `get` function in VAR objects are also accessible in SVAR objects.

### Valid queries on SVAR objects

### Description

### Example

---

## ■ sort

Sort SVAR parameterisations by squared distance of shock responses to median

### Syntax

```
[B,DATA,INDEX,CRIT] = sort(A,DATA,SORTBY,...)
```

### Input arguments

- `A [ SVAR ]` - SVAR object with multiple parameterisations that will be sorted.

- DATA [ tseries | struct | empty ] - SVAR data (endogenous variables and structural shocks); the structural shocks will be re-ordered according to the SVAR parameterisations. You can leave the input argument DATA empty.
- SORTBY [ char ] - Text string that will be evaluated to compute the criterion by which the parameterisations will be sorted; see Description for how to write SORTBY.

### Output arguments

- B [ SVAR ] - SVAR object with parameterisations sorted by the specified criterion.
- DATA [ tseries | struct | empty ] - SVAR data with the structural shocks re-ordered to correspond to the order of parameterisations.
- INDEX [ numeric ] - Vector of indices so that  $B = A(\text{INDEX})$ .
- CRIT [ numeric ] - The value of the criterion based on the string SORTBY for each parameterisation.

### Options

- 'progress=' [ true | false ] - Display progress bar in the command window.

### Description

The individual parameterisations within the SVAR object A are sorted by the sum squared distances of selected shock responses to the respective median responses. Formally, the following criterion is evaluated for each parameterisation

$$\sum_{i \in I, j \in J, k \in K} [S_{i,j}(k) - M_{i,j}(k)]^2$$

where  $S_{i,j}(k)$  denotes the response of the i-th variable to the j-th shock in period k, and  $M_{i,j}(k)$  is the median responses. The sets of variables, shocks and periods, i.e. I, J, K, respectively, over which the summation runs are determined by the user in the SORTBY string.

How do you select the shock responses that enter the criterion in SORTBY? The input argument SORTBY is a text string that refers to array S, whose element  $S(i,j,k)$  is the response of the i-th variable to the j-th shock in period k.

**Example**

Sort the parameterisations by squared distance to median of shock responses of all variables to the first shock in the first four periods. The parameterisation that is closest to the median responses

```
S2 = sort(S1,[],'S(:,1,1:4)')
```

**■ sprintf**

Format SVAR as a model code and write to text string

**Syntax**

```
[C,D] = sprintf(S,...)
```

**Input arguments**

- `S` [ SVAR ] - SVAR object that will be written as a model code.
- Output arguments
- `C` [ cellstr ] - Text string with the model code for each parameterisation.
- `D` [ cell ] - Parameter database for each parameterisation; if `'hardParameters='` is true, the databases will be empty.

**Options**

- `'decimal='` [ numeric | empty ] - Precision (number of decimals) at which the coefficients will be written if `'hardParameters='` is true; if empty, the `'format='` options is used.
- `'declare='` [ true | false ] - Add declaration blocks and keywords for VAR variables, shocks, and equations.
- `'eNames='` [ cellstr | char | empty ] - Names that will be given to the VAR residuals; if empty, the names from the SVAR object will be used.
- `'format='` [ char | '%+.16e' ] - Numeric format for parameter values; it will be used only if `'decimal='` is empty.

- 'hardParameters=' [ true | false ] - Print coefficients as hard numbers; otherwise, create parameter names and return a parameter database.
- 'yNames=' [ cellstr | char | empty ] - Names that will be given to the variables; if empty, the names from the SVAR object will be used.
- 'tolerance=' [ numeric | getrealsmall() ] - Treat VAR coefficients smaller than 'tolerance=' in absolute value as zeros; zero coefficients will be dropped from the model code.

## Description

## Example

---

## ■ srf

### Shock (impulse) response function

#### Syntax

```
[R,CUM] = srf(V,NPER)
[R,CUM] = srf(V,RANGE)
```

#### Input arguments

- V [ SVAR ] - SVAR object for which the impulse response function will be computed.
- NPER [ numeric ] - Number of periods.
- RANGE [ numeric ] - Date range.

#### Output arguments

- R [ tseries | struct ] - Shock response functions.
- CUM [ tseries | struct ] - Cumulative shock response functions.

### Options

- 'presample=' [ true | false ] - Include zeros for pre-sample initial conditions in the output data.
- 'select=' [ cellstr | char | logical | numeric | Inf ] - Selection of shocks to which the responses will be simulated.

### Description

For backward compatibility, the following calls into the srf function is also possible:

```
[~,~,s,c] = srf(this,nper)
[~,~,s,c] = srf(this,range)
```

### Example

---

## ■ SVAR

Convert reduced-form VAR to structural VAR

### Syntax

```
[S,DATA,B,COUNT] = SVAR(V,DATA,...)
```

### Input arguments

- V [ VAR ] - Reduced-form VAR object.
- DATA [ struct | tseries ] - Data associated with the input VAR object.

### Output arguments

- S [ VAR ] - Structural VAR object.
- DATA [ struct | tseries ] - Data with transformed structural residuals.



- `B` [ numeric ] - Impact matrix of structural residuals.
- `COUNT` [ numeric ] - Number of draws actually performed (both successful and unsuccessful) when `'method'='draw'`; otherwise `COUNT=1`.

## Options

- `'maxIter='` [ numeric | 0 ] - Maximum number of attempts when `'method'='draw'`.
- `'method='` [ 'chol' | `'draw'` | `'qr'` | `'svd'` ] - Method that will be used to identify structural residuals.
- `'nDraw='` [ numeric | 0 ] - Target number of successful draws when `'method'='draw'`.
- `'output='` [ 'auto' | `'dbase'` | `'tseries'` ] - Format of output data.
- `'progress='` [ `true` | `false` ] - Display progress bar in the command window.
- `'rank='` [ numeric | Inf ] - Reduced rank of the covariance matrix of structural residuals when `'method=' 'svd'`; Inf means full rank is preserved.
- `'std='` [ numeric | 1 ] - Std deviation of structural residuals.
- `'test='` [ char ] - String that will be evaluated for each random draw of the impact matrix B. The evaluation must result in true or false; only those matrices B that evaluate to true will be kept. See Description for more on how to write test.
- `'ordering='` [ numeric | empty ] - Re-order VAR variables before identifying structural residuals.
- `'reorderResiduals='` [ `true` | `false` ] - Re-order identified structural residuals back.

## Description

—Identification random Householder transformations

The structural impact matrices B are randomly generated using a Householder transformation algorithm. Each matrix is tested by evaluating the test string supplied by the user. If it evaluates to true the matrix is kept and one more SVAR parameterisation is created, if it is false the matrix is discarded.

The test string can refer to the following characteristics:

- `S` — the impulse (or shock) response function; the `S(i,j,k)` element is the response of the *i*-th variable to the *j*-th shock in period *k*.
- `Y` — the asymptotic cumulative response function; the `Y(i,j)` element is the asymptotic (long-run) cumulative response of the *i*-th variable to the *j*-th shock.

### **Example**

## 14 Bayesian VAR prior dummies: BVAR package

The BVAR package can be used to create the basic types of prior dummy observations when estimating Bayesian VAR models. The dummy observations are passed in the [VAR/estimate](#) P179 function through the 'BVAR=' option.

### Constructing dummy observations

- [litterman](#) P211 - Litterman's prior dummy observations for BVARs.
- [sumofcoeff](#) P212 - Doan et al sum-of-coefficient prior dummy observations for BVARs.
- [uncmean](#) P213 - Unconditional-mean dummy (or Sims' initial dummy) observations for BVARs.

### Getting help on BVAR functions

```
help BVAR
help BVAR/function_name
```

## ■ litterman

Litterman's prior dummy observations for BVARs

### Syntax

```
[X,Y0,K,Y1,G1] = BVAR.litterman(RHO,SIGMA,LAMBDA,[NY,P])
[X,Y0,K,Y1,G1] = BVAR.litterman(RHO,SIGMA,LAMBDA,[NY,P,NG])
```

### Input arguments

- RHO [ numeric ] - White-noise priors (RHO = 0) or random-walk priors (RHO = 1).
- SIGMA [ numeric ] - Weight on the dummy observations.
- LAMBDA [ numeric ] - Exponential increase in weight depending on the lag; LAMBDA = 0 means all lags are weighted equally.
- [NY,P,NG] [ numeric ] - Number of variables, order of the VAR, and number of co-integrating vectors in the VAR for which the prior dummies will be created.

### Output arguments

- `X` [ numeric ] - Array with prior dummy observations that can be used in the 'BVAR=' option of the [VAR/estimate](#) P179 function.
- `Y0, K, Y1, G1` [ numeric ] - These extra output arguments only provide more details on the structure of the dummy observations, and have no use; `X=[Y0;K;Y1;G1]`.

### Description

### Example

---

## ■ sumofcoeff

Doan et al sum-of-coefficient prior dummy observations for BVARs

### Syntax

```
[X,Y0,K,Y1,G1] = BVAR.sumofcoeff(MU,[NY,P])  
[X,Y0,K,Y1,G1] = BVAR.sumofcoeff(MU,[NY,P,NG])
```

### Input arguments

- `MU` [ numeric ] - Weight on the dummy observations.
- `[NY,P,NG]` [ numeric ] - Number of variables, order of the VAR, and number of co-integrating vectors in the VAR for which the prior dummies will be created.

### Output arguments

- `X` [ numeric ] - Array with prior dummy observations that can be used in the 'BVAR=' option of the [VAR/estimate](#) P179 function.
- `Y0, K, Y1, G1` [ numeric ] - These extra output arguments only provide more details on the structure of the dummy observations, and have no use; `X=[Y0;K;Y1;G1]`.

## Description

## Example

---

## ■ uncmean

Unconditional-mean dummy (or Sims' initial dummy) observations for BVARs

## Syntax

```
[X,Y0,K,Y1,G1] = BVAR.uncmean(YBAR,MU,[NY,P])
[X,Y0,K,Y1,G1] = BVAR.uncmean(ybar,mu,[ny,p,ng])
```

## Input arguments

- YBAR [ numeric ] - Vector of unconditional means imposed as priors.
- MU [ numeric ] - Weight on the dummy observations.
- [NY,P,NG] [ numeric ] - Number of variables, order of the VAR, and number of co-integrating vectors in the VAR for which the prior dummies will be created.

## Output arguments

- X [ numeric ] - Array with prior dummy observations that can be used in the 'BVAR=' option of the [VAR/estimate](#) P179 function.
- Y0, K, Y1, G1 [ numeric ] - These output arguments provide more details on the structure of the dummy observations, and have no use; X=[Y0;K;Y1;G1].

## Description

## Example

## 15 Factor-augmented vector autoregressions: FAVAR objects and functions

### Constructor

- [FAVAR](#) P217 - Create new FAVAR object.

### Getting information about FAVAR objects

- [comment](#) P214 - Get or set user comments in an IRIS object.
- [get](#) P220 - Query model object properties.
- [userdata](#) P222 - Get or set user data in an IRIS object.
- [VAR](#) P222 - Return a VAR object describing the factor dynamics.

### Estimation

- [estimate](#) P215 - Estimate FAVAR using static principal components.

### Filtering and forecasting

- [filter](#) P218 - Re-estimate the factors by Kalman filtering the data taking FAVAR coefficients as given.
- [forecast](#) P219 - Forecast FAVAR factors and observables.

### Getting on-line help on FAVAR functions

```
help FAVAR
help FAVAR/function_name
```

---

## ■ comment

Get or set user comments in an IRIS object

### Syntax for getting user comments

```
C = comment(OBJ)
```

### Syntax for assigning user comments

```
OBJ = comment(OBJ,C)
```

### Input arguments

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

### Output arguments

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

### Description

### Example

---

## ■ estimate

Estimate FAVAR using static principal components

### Syntax

```
[A,D,CC,F,U,E,CTF] = estimate(A,D,LIST,RANGE,[R,Q],...)  
[A,X,CC,F,U,E,CTF] = estimate(A,X,RANGE,[R,Q],...)
```

### Input arguments

- A [ FAVAR ] - Empty FAVAR object.

- `D [ struct ]` - Input database.
- `LIST [ cellstr ]` - List of series from the database on which the VAR will be estimated.
- `X [ tseries ]` - Tseries objects with input data.
- `RANGE [ numeric ]` - Estimation range.
- `R [ numeric ]` - Selection criterion for the number of factors: Minimum requested roportion of input data volatility explained by the factors.
- `Q [ numeric ]` - Selection criterion for the number of factors: Maximum number of factors.

### Output arguments

- `A [ FAVAR ]` - Estimated FAVAR object.
- `D [ struct ]` - Output database.
- `X [ tseries ]` - Output tseries object.
- `CC [ tseries ]` - Estimates of common components in the FAVAR observables.
- `F [ tseries ]` - Estimates of factors.
- `U [ struct | tseries ]` - Idiosyncratic residuals.
- `E [ tseries ]` - Factor VAR residuals.
- `CTF [ tseries ]` - Contributions of individual input series to the estimated factors.

### Options

- `'cross=' [ true | false | numeric ]` - Keep off-diagonal elements in the covariance matrix of idiosyncratic residuals; if false all cross-covariances are reset to zero; if a number between zero and one, all cross-covariances are multiplied by that number.
- `'order=' [ numeric | 1 ]` - Order of the VAR for factors.
- `'output=' [ 'auto' | 'dbase' | 'tseries' ]` - Format of output data.
- `'rank=' [ numeric | Inf ]` - Restriction on the rank of the factor VAR residuals.



## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ FAVAR

Create new FAVAR object

## Syntax

```
f = FAVAR()
```

## Output arguments

- `f [ FAVAR ]` - New empty VAR object.

## Description

This function creates a new empty FAVAR object. It is usually followed by the [estimate](#) P215 function to estimate the FAVAR parameters on data.

## Example

To estimate a FAVAR, you first need to create an empty VAR object, and then run the [FAVAR](#) P215 function on it, e.g.

```
f = FAVAR();  
f = estimate(f,d,list,range);
```

or simply

```
f = estimate(VAR(),d,list,range);
```

---

## ■ filter

Re-estimate the factors by Kalman filtering the data taking FAVAR coefficients as given

### Syntax

```
[a,d,cc,f,u,e] = filter(a,d,range,...)
```

### Input arguments

- `f` [ FAVAR ] - Estimated FAVAR object.
- `d` [ struct | tseries ] - Input database or tseries object with the FAVAR observables.
- `range` [ numeric ] - Filter date range.

### Output arguments

- `a` [ FAVAR ] - FAVAR object.
- `d` [ struct ] - Output database or tseries object with the FAVAR observables.
- `cc` [ struct | tseries ] - Re-estimated common components in the observables.
- `f` [ tseries ] - Re-estimated common factors.
- `u` [ tseries ] - Re-estimated idiosyncratic residuals.
- `e` [ tseries ] - Re-estimated structural residuals.

### Options

- `'cross='` [ true | false | numeric ] - Run the filter with the off-diagonal elements in the covariance matrix of idiosyncratic residuals; if false all cross-covariances are reset to zero; if a number between zero and one, all cross-covariances are multiplied by that number.
- `'invFunc='` [ 'auto' | function\_handle ] - Inversion method for the FMSE matrices.
- `'meanOnly='` [ true | false ] - Return only mean data, i.e. point estimates.
- `'persist='` [ true | false ] - If filter or forecast is used with 'persist=' set to true for the first time, the forecast MSE matrices and their inverses will be stored; subsequent calls of the filter or forecast functions will re-use these matrices until filter or forecast is called.

- 'output=' [ 'auto' | 'dbase' | 'tseries' ] - Format of output data.
- 'tolerance=' [ numeric | 0 ] - Numerical tolerance under which two FMSE matrices computed in two consecutive periods will be treated as equal and their inversions will be re-used, not re-computed.

### Description

It is the user's responsibility to make sure that filter and forecast called with 'persist=' set to true are valid, i.e. that the previously computed FMSE matrices can be really re-used in the current run.

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ forecast

Forecast FAVAR factors and observables

### Syntax

```
[D,CC,F,U,E] = forecast(A,D,RANGE,J,...)
```

### Input arguments

- A [ FAVAR ] - FAVAR object.
- D [ struct | tseries ] - Input data with initial condition for the FAVAR factors.
- RANGE [ numeric ] - Forecast range.
- J [ struct | tseries ] - Conditioning data with hard tunes on the FAVAR observables.

### Output arguments

- D [ struct ] - Output database or tseries object with the FAVAR observables.
- CC [ struct | tseries ] - Projection of common components in the observables.

Factor-augmented vector autoregressions: FAVAR objects and functions: `get`

- `F [ tseries ]` - Projection of common factors.
- `U [ tseries ]` - Conditional idiosyncratic residuals.
- `E [ tseries ]` - Conditional structural residuals.

## Options

See help on [FAVAR/filter](#) P218 for options available.

## Description

## Example

---

# ■ `get`

Query model object properties

## Syntax

```
value = get(a,query)
[value,value,...] = get(a,query,query,...)
```

## Input arguments

- `a [ FAVAR ]` - FAVAR object.
- `query [ char ]` - Name of the queried property.

## Output arguments

- `value [ ... ]` - Value of the queried property.

## Valid queries on FAVAR objects

*System matrices*

- 'A\*' Returns [ numeric ] the transition matrix of the underlying VAR system on factors.
- 'B' Returns [ numeric ] the matrix mapping the impact of structural residuals on the factors in the underlying VAR.
- 'C' Returns [ numeric ] the matrix mapping the factors into the observables.
- 'Omega' Returns [ numeric ] the reduced-form covariance matrix of the residuals in the underlying VAR.
- 'Sigma' Returns [ numeric ] the covariance matrix of idiosyncratic shocks.

#### *Underlying VAR*

- 'VAR=' Returns [ VAR ] a VAR object describing the factor dynamics.

#### *Eigenvalues and singular values*

- 'eig' Returns [ numeric ] the vector of eigenvalues of the underlying VAR.
- 'sing' Returns [ numeric ] the vector of singular values from the principal component estimation step.

#### *Observables and factors*

- 'mean' Returns [ numeric ] the estimated mean of the observables used to standardise the input data.
- 'std' Returns [ numeric ] the estimated std deviations of the observables used to standardise the input data.
- 'ny' Returns [ numeric ] the number of observables.
- 'nx' Returns [ numeric ] the number of factors.
- 'yList' Returns [ cellstr ] the list of the names of observables.

### **Description**

### **Example**

## ■ userdata

Get or set user data in an IRIS object

Syntax for getting user data

```
X = userdata(OBJ)
```

Syntax for assigning user data

```
OBJ = userdata(OBJ,X)
```

Input arguments

- OBJ [ model | tseries | VAR | SVAR | FAVAR | sstate ] - One of the IRIS objects.
- X [ ... ] - Data that will be attached to the object.

Output arguments

- X [ ... ] - User data that are currently attached to the object.

Description

Example

---

## ■ VAR

Return a VAR object describing the factor dynamics

Syntax

```
v = VAR(a)
```

### **Input arguments**

a [ FAVAR ] - FAVAR object.

### **Output arguments**

v [ VAR ] - VAR object describing the dynamic system of the FAVAR factors.

### **Description**

### **Example**

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

Part IV —  
Time series and database management



## 16 Dates and date ranges

### Creating IRIS serial date numbers

- `bb` P226 - IRIS serial date numbers for dates with bi-monthly frequency.
- `bbtoday` P227 - IRIS serial date number for current bi-month.
- `hh` P240 - IRIS serial date numbers for dates with half-yearly frequency.
- `hhtoday` P241 - IRIS serial date number for current half-year.
- `mm` P241 - IRIS serial date numbers for dates with monthly frequency.
- `mmtoday` P242 - IRIS serial date number for current month.
- `qq` P242 - IRIS serial date numbers for dates with quarterly frequency.
- `qqtoday` P243 - IRIS serial date number for current quarter.
- `yy` P245 - IRIS serial date numbers for dates with yearly frequency.
- `yytoday` P246 - IRIS serial date number for current year.

### Computing special dates (daily dates only)

- `datbom` P233 - Beginning of month for the specified daily date.
- `datboq` P234 - Beginning of quarter for the specified daily date.
- `datboy` P234 - Beginning of year for the specified daily date.
- `dateom` P237 - End of month for the specified daily date.
- `dateoq` P238 - End of quarter for the specified daily date.
- `dateoy` P238 - End of year for the specified daily date.

### Creating date ranges

- `daterange` P239 - Use the colon operator to create date ranges.

## Converting dates

- [clp2dat](#) P227 - Convert text in system clipboard to dates.
- [dat2char](#) P228 - Convert dates to character array.
- [dat2charlist](#) P229 - Convert dates to a comma-separated list.
- [dat2clp](#) P230 - Convert dates to text and paste to system clipboard.
- [dat2dec](#) P231 - Convert dates to their decimal representations.
- [dat2str](#) P231 - Convert IRIS dates to cell array of strings.
- [dat2ypf](#) P233 - Convert IRIS serial date number to year, period and frequency.
- [dec2dat](#) P239 - Convert decimal numbers to IRIS serial date numbers.
- [str2dat](#) P244 - Convert strings to IRIS serial date numbers.

## Date comparison

- [datcmp](#) P235 - Compare two IRIS serial date numbers.
- [datdiff](#) P236 - Number of periods between two dates with check for date frequency.
- [rngcmp](#) P243 - Compare two IRIS date ranges.

## Getting on-line help on date functions

```
help dates
help dates/function_name
```

---

## ■ bb

IRIS serial date numbers for dates with bi-monthly frequency

### Syntax

```
d = bb(y)
d = bb(y,b)
```

### Input arguments

- y [ numeric ] - Years.
- q [ numeric ] - B-months; if missing, first bi-month is assumed.

### Output arguments

- d [ numeric ] - IRIS serial date numbers representing the input bi-months.

### Description

### Example

---

## ■ bbtoday

IRIS serial date number for current bi-month

### Syntax

```
d = bbtoday()
```

### Output arguments

- d [ numeric ] - IRIS serial date number for current bi-month.

### Description

### Example

---

## ■ clp2dat

Convert text in system clipboard to dates

### Syntax

```
D = clp2dat(...)
```

### Output arguments

- D [ numeric ] - IRIS serial date numbers based on the current content of the system clipboard converted by the [str2dat](#) P244 function.

### Options

See help on [str2dat](#) P244 for options available.

### Description

### Example

---

## ■ dat2char

Convert dates to character array

### Syntax

```
C = dat2char(D,...)
```

### Input arguments

- D [ numeric ] - IRIS serial date numbers that will be converted to character array.

### Output arguments

- C [ char ] - Character array representing the input dates; each line of the array represents one date from D.

## Options

See help on [dat2str](#) P231 for options available.

## Description

### Example

We create a quarterly date using the function `qq`; this function returns an IRIS serial date number. We then use `dat2char` to print a human-readable text representation of that date.

```
d = qq(2015,3)
d =
    8.0620e+03
dat2char(d)
ans =
    2015Q3
```

---

## ■ dat2charlist

Convert dates to a comma-separated list

### Syntax

```
C = dat2charlist(D,...)
```

### Input arguments

- `D [ numeric ]` - IRIS serial date numbers that will be converted to a comma-separated list.

### Output arguments

- `C [ char ]` - Text string with a comma-separated list of dates.

### Options

See help on [dat2str](#) P231 for options available.

### Description

### Example

---

## ■ dat2clp

Convert dates to text and paste to system clipboard

### Syntax

```
C = dat2clp(D,...)
```

### Input arguments

- D [ numeric ] - IRIS serial date numbers that will be converted to character array and pasted to the system clipboard.

### Output arguments

- C [ char ] - Character array representing the input dates pasted to the system clipboard; each line of the array represents one date from D.

### Options

See help on [dat2str](#) P231 for options available.

### Description

### Example

---

## ■ dat2dec

Convert dates to their decimal representations

### Syntax

```
DEC = dat2dec(DAT)
```

### Input arguments

- DAT [ numeric ] - IRIS serial date number.

### Output arguments

- DEC [ numeric ] - Decimal number representing the input dates, computed as  $\text{year} + (\text{per}-1)/\text{freq}$ .

### Description

### Example

---

## ■ dat2str

Convert IRIS dates to cell array of strings

### Syntax

```
s = dat2str(dat,...)
```

### Input arguments

- dat [ numeric ] - IRIS serial date number(s).

### Output arguments

- s [ cellstr ] - Cellstr with strings representing the input dates.

## Options

- 'dateFormat=' [ char | 'YYYYFP' ] - Date format string.
- 'freqLetters=' [ char | 'YHQBM' ] - Letters representing the five possible frequencies (annual,semi-annual,quarterly,bimontly,monthly).
- 'months=' [ cellstr | English names of months ] - Strings representing the twelve months.
- 'standinMonth=' [ numeric | 'last' | 1 ] - Which month will represent a lower-than-monthly-frequency date if month is part of the date format string.

## Description

The date format string can include any combination of the following fields:

- 'Y=' - Year.
- 'YYYY=' - Four-digit year.
- 'YY=' - Two-digit year.
- 'P=' - Period within the year (half-year, quarter, bi-month, month).
- 'PP=' - Two-digit period within the year.
- 'R=' - Upper-case roman numeral for the period within the year.
- 'r=' - Lower-case roman numeral for the period within the year.
- 'M=' - Month numeral.
- 'MM=' - Two-digit month numeral.
- 'MMMM=', 'Mmmm', 'mmm' - Case-sensitive name of month.
- 'MMM=', 'Mmm', 'mmm' - Case-sensitive three-letter abbreviation of month.
- 'F=' - Upper-case letter representing the date frequency.
- 'f=' - Lower-case letter representing the date frequency.

To get some of the above letters printed literally in the date string, use a percent sign as an escape character, i.e. '%Y', etc.

## Example



## ■ dat2ypf

Convert IRIS serial date number to year, period and frequency

### Syntax

```
[y,p,f] = dat2ypf(dat)
```

### Input arguments

- dat [ numeric ] - IRIS serial date numbers.

### Output arguments

- y [ numeric ] - Years.
- p [ numeric ] - Periods within year.
- f [ numeric ] - Date frequencies.

### Description

### Example

---

## ■ datbom

Beginning of month for the specified daily date

### Syntax

```
BOM = datebom(D)
```

### Input arguments

- D [ numeric ] - Daily serial date number.

### Output arguments

- BOM [ numeric ] - Daily serial date number for the first day of the same month as D.

### Description

### Example

---

## ■ datboq

Beginning of quarter for the specified daily date

### Syntax

BOQ = datboq(D)

### Input arguments

- D [ numeric ] - Daily serial date number.

### Output arguments

- BOQ [ numeric ] - Daily serial date number for the first day of the same quarter as D.

### Description

### Example

---

## ■ datboy

Beginning of year for the specified daily date

### Syntax

```
BOY = dateboy(D)
```

### Input arguments

- `D` [ numeric ] - Daily serial date number.

### Output arguments

- `BOY` [ numeric ] - Daily serial date number for the first day of the same year as `D`.

### Description

### Example

---

## ■ `datecmp`

Compare two IRIS serial date numbers

### Syntax

```
FLAG = datecmp(D1,D2)
```

### Input arguments

- `D1, D2` [ numeric ] - IRIS serial date numbers or vectors.

### Output arguments

- `FLAG` [ true | false ] - True for numbers that represent the same date.

## Description

The two date vectors must either be the same lengths, or one of them must be scalar.

Use this function instead of the plain comparison operator, `==`, to compare dates. The plain comparison can sometimes give false results because of round-off errors.

## Example

```
d1 = qq(2010,1);
d2 = qq(2009,1):qq(2010,4);
datcmp(d1,d2)
ans =
    0    0    0    0    1    0    0    0
```

---

## ■ datdiff

Number of periods between two dates with check for date frequency

## Syntax

```
D = datdiff(D1,D2)
```

## Input arguments

- D1, D2 [ numeric ] - IRIS dates or vectors of IRIS dates.

## Output arguments

- D [ numeric ] - Number of periods between D1 and D2, positive for D1 greater than D2, negative for D1 smaller than D2, or NaN for dates of different frequencies.

## Description

## Example

```
d1 = mm(2010,12);
```

```
d2 = mm(2011,12);
```

```
datdiff(d1,d2)
```

```
ans =  
    -12
```

```
datdiff(d2,d1)
```

```
ans =  
     12
```

```
d3 = yy(2011);
```

```
datdiff(d1,d3)
```

```
ans =  
    NaN
```

---

## ■ dateom

End of month for the specified daily date

### Syntax

```
EOM = dateom(D)
```

### Input arguments

- D [ numeric ] - Daily serial date number.

### Output arguments

- EOM [ numeric ] - Daily serial date number for the last day of the same month as D.

### Description

### Example

---

## ■ dateoq

End of quarter for the specified daily date

### Syntax

EQQ = dateoq(D)

### Input arguments

- D [ numeric ] - Daily serial date number.

### Output arguments

- EQQ [ numeric ] - Daily serial date number for the last day of the same quarter as D.

### Description

### Example

---

## ■ dateoy

End of year for the specified daily date

### Syntax

EOY = dateoy(D)

### Input arguments

- D [ numeric ] - Daily serial date number.

### Output arguments

- EOY [ numeric ] - Daily serial date number for the last day of the same year as D.

## Description

## Example

---

## ■ daterange

Use the colon operator to create date ranges

## Syntax

```
startdate : enddate  
startdate : increment : enddate
```

## Input arguments

- startdate [ numeric ] - IRIS serial date number representing the startdate.
- enddate [ numeric ] - IRIS serial date number representing the enddate; startdate and enddate must be the same frequency.
- increment [ numeric ] - Number of periods (specific to each frequency) between the dates in the date vector.

## Description

You can use the colon operator to create continuous date ranges because the IRIS serial date numbers are designed so that whatever the frequency two consecutive dates are represented by numbers that differ exactly by one.

## Example

---

## ■ dec2dat

Convert decimal numbers to IRIS serial date numbers

### Syntax

```
dat = dec2dat(dec,freq)
```

### Input arguments

- dec [ numeric ] - Decimal numbers representing dates.
- freq [ freq ] - Date frequency.

### Output arguments

- dat [ numeric ] - IRIS serial data numbers corresponding to the input decimals.
- 

## ■ hh

IRIS serial date numbers for dates with half-yearly frequency

### Syntax

```
d = hh(y)  
d = hh(y,h)
```

### Input arguments

- y [ numeric ] - Years.
- h [ numeric ] - Half-years; if missing, first half-year is used.

### Output arguments

- d [ numeric ] - IRIS serial date numbers representing the input half-years.



### Description

### Example

---

## ■ hhtoday

IRIS serial date number for current half-year

### Syntax

```
d = hhtoday()
```

### Output arguments

- d [ numeric ] - IRIS serial date number for current half-year.

### Description

### Example

---

## ■ mm

IRIS serial date numbers for dates with monthly frequency

### Syntax

```
d = mm(y)
d = mm(y,m)
```

### Input arguments

- y [ numeric ] - Years.
- m [ numeric ] - Months; if missing, first month (January) is assumed.

### Output arguments

- d [ numeric ] - IRIS serial date numbers representing the input months.

### Description

### Example

---

## ■ mmtoday

IRIS serial date number for current month

### Syntax

```
d = mmtoday()
```

### Output arguments

- d [ numeric ] - IRIS serial date number for current month.

### Description

### Example

---

## ■ qq

IRIS serial date numbers for dates with quarterly frequency

### Syntax

```
d = qq(y)
d = qq(y,q)
```

### Input arguments

- y [ numeric ] - Years.
- q [ numeric ] - Quarters; if missing, first quarter is assumed.

### Output arguments

- d [ numeric ] - IRIS serial date numbers representing the input quarterly dates.

### Description

### Example

---

## ■ qtoday

IRIS serial date number for current quarter

### Syntax

```
d = qtoday()
```

### Output arguments

- d [ numeric ] - IRIS serial date number for current quarter.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ rngcmp

Compare two IRIS date ranges

## Syntax

```
FLAG = rngcmp(R1,R2)
```

## Input arguments

- R1, R2 [ numeric ] - IRIS date ranges.

## Output arguments

- FLAG [ true | false ] - True if the two date ranges are the same.

## Description

An IRIS date range is distinct from a vector of dates in that only the first and the last dates matter. Often, date ranges are context sensitive. In that case, you can use `-Inf` for the start date (meaning the earliest possible date in the given context) and `Inf` for the end date (meaning the latest possible date in the given context), or simply `Inf` for the whole range (meaning from the earliest possible date to the latest possible date in the given context).

## Example

```
r1 = qq(2010,1):qq(2020,4);
r2 = [qq(2010,1),qq(2020,4)];

rngcmp(r1,r2)
ans =
     1
```

---

## ■ str2dat

Convert strings to IRIS serial date numbers

## Syntax

```
DAT = str2dat(S,...)
```

### Input arguments

- S [ char | cellstr ] - Strings representing dates.

### Output arguments

- DAT [ numeric ] - IRIS serial date numbers.

### Options

- 'freq=' [ 1 | 2 | 4 | 6 | 12 | empty ] - Enforce frequency.

See help on [dat2str](#) P231 for other options available.

### Description

#### Example

```
d = str2dat('04-2010','dateformat','MM-YYYY');
dat2str(d)
ans =
    '2010M04'

d = str2dat('04-2010','dateformat','MM-YYYY','freq',4);
dat2str(d)
ans =
    '2010Q2'
```

---

## ■ yy

IRIS serial date numbers for dates with yearly frequency

### Syntax

```
d = yy(y)
```

### Input arguments

- y [ numeric ] - Years.

### Output arguments

- d [ numeric ] - IRIS serial date numbers representing the input years.

### Description

### Example

---

## ■ yytoday

IRIS serial date number for current year

### Syntax

```
d = yytoday()
```

### Output arguments

- d [ numeric ] - IRIS serial date number for current year.

### Description

### Example

## 17 Time series objects and functions

Tseries methods:

### Constructor

- `tseries` [P303](#) - Create new time series (tseries) object.

### Getting information about tseries objects

- `daily` [P264](#) - Calendar view of a daily tseries object.
- `enddate` [P269](#) - Date of the last available observation in a tseries object.
- `freq` [P273](#) - Frequency of a tseries object.
- `get` [P274](#) - Query tseries object property.
- `length` [P278](#) - Length of tseries object.
- `ndims` [P282](#) - Number of dimensions in tseries object data.
- `size` [P296](#) - Size of tseries object data.
- `startdate` [P298](#) - Date of the first available observation in a tseries object.
- `yearly` [P310](#) - Display tseries object one full year per row.

### Referencing tseries objects

- `subsasgn` [P301](#) - Subscripted assignment for tseries objects.
- `subsref` [P302](#) - Subscripted reference function for tseries objects.

### Maths and statistics functions and operators

Some of the following functions require the Statistics Toolbox.

+, -, \*, \, /, ^, &, |, ~, ==, ~=, >=, >, <, <=, abs, acos, asin, atan, atan2, ceil, cos, exp, floor, imag, isinf, isnan, log, log10, real, round, sin, sqrt, tan, normpdf, normcdf, prctile, lognpdf, logncdf

The behaviour of the following functions depend on the dimension along which they are performed.

Some of the following functions require the Statistics Toolbox.

all, any, cumprod, cumsum, find, geomean, max, mean, median, min, mode, nanmean, nanstd, nansum, nanvar, prod, std, sum, var

## Filters

- [arf](#) P253 - Run autoregressive function on a tseries object.
- [bpass](#) P256 - Band-pass filter.
- [bwf](#) P259 - Butterworth filter with tunes.
- [bwf2](#) P260 - Swap output arguments of the Butterworth filter with tunes.
- [detrend](#) P266 - Remove a linear time trend.
- [expsmooth](#) P271 - Exponential smoothing.
- [hpf](#) P275 - Hodrick-Prescott filter with tunes.
- [hpf2](#) P277 - Swap output arguments of the Hodrick-Prescott filter with tunes.
- [fft](#) P272 - Discrete Fourier transform of tseries object.
- [llf](#) P279 - Local linear trend (or random-walk-plus-noise) filter with tunes.
- [llf2](#) P281 - Swap output arguments of the local linear trend filter with tunes.
- [moving](#) P281 - Apply function to moving window of observations.
- [trend](#) P303 - Estimate a time trend.
- [x12](#) P306 - Access to X12 seasonal adjustment program.

## Estimation and sample characteristics

Note that most of the sample characteristics are listed above in the Maths and statistics functions and operators section.

- [acf](#) P250 - Sample autocovariance and autocorrelation functions.
- [hpd](#) P275 - Highest probability density interval.
- [chowlin](#) P260 - Chow-Lin distribution of low-frequency observations over higher-frequency periods.
- [regress](#) P290 - Ordinary or weighted least-square regression.



### Visualising tseries objects

- [area](#) P252 - Area graph for tseries objects.
- [bar](#) P254 - Bar graph for tseries objects.
- [barcon](#) P255 - Contribution bar graph for tseries objects.
- [errorbar](#) P270 - Line plot with error bars.
- [plot](#) P285 - Line graph for tseries objects.
- [plotcmp](#) P286 - Comparison graph for two time series.
- [plotpred](#) P287 - Plot Kalman filter predictions.
- [plotyy](#) P288 - Line plot function with LHS and RHS axes for time series.
- [scatter](#) P294 - Scatter graph for tseries objects.
- [spy](#) P297 - Visualise tseries observations that pass a test.
- [stem](#) P300 - Plot tseries as discrete sequence data.

### Manipulating tseries objects

- [empty](#) P268 - Empty tseries object preserving its size in 2nd and higher dimensions.
- [permute](#) P284 - Permute dimensions of a tseries object.
- [redate](#) P289 - Change time dimension of a tseries object.
- [reshape](#) P291 - Reshape size of time series in 2nd and higher dimensions.
- [resize](#) P292 - Clip tseries object down to specified date range.
- [sort](#) P297 - Sort tseries columns by specified criterion.

### Converting tseries objects

- [convert](#) P261 - Convert tseries object to a different frequency.
- [double](#) P267 - Return tseries observations as double-precision numeric array.
- [doubledata](#) P268 - Convert tseries observations to double precision.
- [single](#) P295 - Return tseries observations as single-precision numeric array.
- [singledata](#) P295 - Convert tseries observations to single precision.

### Other tseries functions

- [apct](#) P251 - Annualised percent rate of change.
- [bsxfun](#) P258 - Standard BSXFUN implemented for tseries objects.
- [cumsumk](#) P263 - Cumulative sum with a k-period leap.
- [destdise](#) P265 - Destandardise tseries object by applying specified standard deviation and mean to it.
- [diff](#) P266 - First difference.
- [interp](#) P278 - Interpolate missing observations.
- [normalise](#) P283 - Normalise data to particular date.
- [pct](#) P284 - Percent rate of change.
- [round](#) P293 - Round tseries data to specified number of decimals.
- [stdise](#) P299 - Standardise tseries data by subtracting mean and dividing by std deviation.
- [windex](#) P304 - Simple weighted or Divisia index.
- [wmean](#) P305 - Weighted average of time series observations.

### Getting on-line help on tseries functions

```
help tseries  
help tseries/function_name
```

---

## ■ acf

### Sample autocovariance and autocorrelation functions

#### Syntax

```
[C,R] = acf(X)  
[C,R] = acf(X,DATES,...)
```

### Input arguments

- `X [ tseries ]` - Tseries object.
- `DATES [ numeric | Inf ]` - Dates or date range on which tseries data will be used.

### Output arguments

- `C [ numeric ]` - Auto-/cross-covariance matrices.
- `R [ numeric ]` - Auto-/cross-correlation matrices.

### Options

- `'demean=' [ true | false ]` - Remove mean from the data before computing the ACF.
- `'order=' [ numeric | 0 ]` - Order up to which the ACF will be computed.
- `'smallSample=' [ true | false ]` - Adjust degrees of freedom for small samples.

### Description

### Example

---

## ■ apct

Annualised percent rate of change

### Syntax

```
x = apct(x)
```

### Input arguments

- `x [ tseries ]` - Input tseries object.

### Output arguments

- `x [ tseries ]` - Annualised percentage rate of change in the input data.

## Description

## Example

---

## ■ area

Area graph for tseries objects

## Syntax

```
[h,range] = area(x,...)
[h,range] = area(range,x,...)
[h,range] = area(a,range,x,...)
```

## Input arguments

- a [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- range [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- x [ tseries ] - Input tseries object whose columns will be plotted as an area graph.

## Output arguments

- h [ numeric ] - Handle(s) to the area(s) plotted.
- range [ numeric ] - Actually plotted date range.

## Options

- 'dateformat=' [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- 'datetick=' [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.

See help on built-in area function for other options available.

## Description

## Example

```
}
```

---

## ■ arf

Run autoregressive function on a tseries object

## Syntax

```
X = arf(X,A,Z,RANGE,...)
```

## Input arguments

- X [ tseries ] - Input data from which initial condition will be taken.
- A [ numeric ] - Vector of coefficients of the autoregressive polynomial.
- Z [ numeric | tseries ] - Exogenous input or constantn in the autoregressive process.
- RANGE [ numeric | Inf ] - Date range on which the new time series observations will be computed; RANGE does not include pre-sample initial condition. Inf means the entire possible range will be used (taking into account the length of pre-sample initial condition needed).

## Output arguments

- X [ tseries ] - Output data with new observations created by running an autoregressive process described by A and Z.

## Description

The autoregressive process has one of the following forms:

$$a_1*x + a_2*x(-1) + \dots + a_n*x(-n) = z,$$

or

$$a_1x + a_2x(+1) + \dots + a_nx(+n) = z,$$

depending on whether the range is increasing (running forward in time), or decreasing (running backward in time). The coefficients  $a_1, \dots, a_n$  are gathered in the A vector,

$$A = [a_1, a_2, \dots, a_n].$$

### Example

The following two lines create an autoregressive process constructed from normally distributed residuals,

$$x_t = \rho x_{t-1} + \epsilon_t$$

```
rho = 0.8;
X = tseries(1:20,@randn);
X = arf(X,[1,-rho],X,2:20);
```

---

## ■ bar

Bar graph for tseries objects

### Syntax

```
[h,range] = bar(x,...)
[h,range] = bar(range,x,...)
[h,range] = bar(a,range,x,...)
```

### Input arguments

- a [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.

- `range [ numeric ]` - Date range; if not specified the entire range of the input tseries object will be plotted.
- `x [ tseries ]` - Input tseries object whose columns will be plotted as a bar graph.

#### Output arguments

- `h [ numeric ]` - Handles to the bars plotted.
- `range [ numeric ]` - Actually plotted date range.

#### Options

- `'dateformat=' [ char | irisget('plotdateformat') ]` - Date format for the tick marks on the x-axis.
- `'datetick=' [ numeric | Inf ]` - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.

See help on built-in bar function for other options available.

#### Description

#### Example

```
}
```

---

## ■ barcon

Contribution bar graph for tseries objects

#### Syntax

```
[h,range] = barcon(x,...)
[h,range] = barcon(range,x,...)
[h,range] = barcon(a,range,x,...)
```

### Input arguments

- `a` [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- `range` [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- `x` [ tseries ] - Input tseries object whose columns will be plotted as a contribution bar graph.

### Output arguments

- `h` [ numeric ] - Handle(s) to the bars plotted.
- `range` [ numeric ] - Actually plotted date range.

### Options

- `'barWidth='` [ numeric | 0.8 ] - Width of bars as a percentage of the space each period occupies on the x-axis.
- `'colorMap='` [ numeric | get(gcf(),'colorMap') ] - Color map used to fill the contribution bars.
- `'dateFormat='` [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- `'dateTick='` [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.
- `'evenlySpread='` [ true | false ] - Colors picked for the contribution bars are evenly spread across the color map.
- `'ordering='` [ 'ascend' | 'descend' | 'preserve' | numeric ] - Ordering of contributions with the same sign within each period; 'preserve' means the original order will be preserved.

### Description

### Example

---

## ■ bpass

Band-pass filter



**Syntax**

```
[X,T] = bpass(X,BAND,RANGE,...)
```

**Output arguments**

- `X` [ `tseries` ] - Band-pass filtered `tseries` object.
- `T` [ `tseries` ] - Estimated trend `tseries` object.

**Input arguments**

- `X` [ `tseries` ] - Input `tseries` object that will be filtered.
- `RANGE` [ `numeric` | `Inf` ] Date range on which the data will be filtered.
- `BAND` [ `numeric` ] - Band of periodicities to be retained in the output data, `BAND` = [`LOW`,`HIGH`].

**Options**

- `'addTrend='` [ `true` | `false` ] - Add the estimated linear time trend back to filtered output series if band includes `Inf`.
- `'detrend='` [ `true` | `false` ] - Remove an estimated time trend from the data before filtering.
- `'log='` [ `true` | `false` ] - Logarithmise the data before filtering, de-logarithmise afterwards.
- `'method='` [ `'cf'` | `'hwfsf'` ] - Type of band-pass filter: Christiano-Fitzgerald, or h-windowed frequency-selective filter.
- `'unitRoot='` [ `true` | `false` ] - Assume unit root in the input data.

See help on [tseries/trend](#) P303 for other options available when `'detrend='` is set to `true`.

**Description**

Christiano, L.J. and T.J.Fitzgerald (2003). The Band Pass Filter. *International Economic Review*, 44(2), 435—465.

Iacobucci, A. & A. Noullez (2005). A Frequency Selective Filter for Short-Length Time Series. *Computational Economics*, 25, 75—102.

## Example

---

### ■ bsxfunc

Standard BSXFUN implemented for tseries objects

#### Syntax

```
Z = bsxfun(FUNC,X,Y)
```

#### Input arguments

- FUNC [ function\_handle ] - Function that will be applied to the input series, FUNC(X,Y).
- X [ tseries | numeric ] - Input time series or numeric array.
- Y [ tseries | numeric ] - Input time series or numeric array.

#### Output arguments

- Z [ tseries ] - Result of FUNC(X,Y) with X and/or Y expanded properly in singleton dimensions.

#### Description

See help on built-in bsxfun for more help.

## Example

We create a multivariate time series and subtract mean from its individual columns.

```
x = tseries(1:10,rand(10,4));  
xx = bsxfun(@minus,x,mean(x));
```

## ■ bwf

Butterworth filter with tunes

### Syntax

```
[T,C] = bwf(X,ORDER,CUTOFF)
[T,C] = bwf(X,ORDER,CUTOFF,RANGE,...)
```

### Syntax with output arguments swapped

```
[C,T] = bwf2(X,ORDER,CUTOFF)
[C,T] = bwf2(X,ORDER,CUTOFF,RANGE,...)
```

### Input arguments

- X [ tseries ] - Input tseries object that will be filtered.
- ORDER [ numeric ] - Order of the Butterworth filter; note that ORDER = 2 reproduces the Hodrick-Prescott filter.
- CUTOFF [ numeric ] - Cut-off periodicity.
- RANGE [ numeric ] - Date range on which the input data will be filtered; RANGE can be Inf, [startdata,Inf], or [-Inf,enddate]; if not specified, Inf (i.e. the entire available range of the input series) is used.

### Output arguments

- T [ tseries ] - Lower-frequency (trend) component.
- C [ tseries ] - Higher-frequency (cyclical) component.

### Options

- 'level=' [ tseries ] - Hard tune on the level of the trend.
- 'change=' [ tseries ] - Hard tune on the change of the trend.
- 'log=' [ true | false ] - Logarithmise the data before filtering, de-logarithmise afterwards.

## Description

## Example

---

### ■ bwf

Swap output arguments of the Butterworth filter with tunes

See help on [tseries/bwf](#) P259.

---

### ■ chowlin

Chow-Lin distribution of low-frequency observations over higher-frequency periods

## Syntax

```
[Y2,B,RHO,U1,U2] = chowlin(Y1,X2)
[Y2,B,RHO,U1,U2] = chowlin(Y1,X2,range,...)
```

## Input arguments

- Y1 [ tseries ] - Low-frequency input tseries object that will be distributed over higher-frequency observations.
- X2 [ tseries ] - Tseries object with regressors used to distribute the input data.
- range [ numeric ] - Low-frequency date range on which the distribution will be computed.

## Output arguments

- Y2 [ tseries ] - Output data distributed with higher frequency.
- B [ numeric ] - Vector of regression coefficients.
- RHO [ numeric ] - Actually used autocorrelation coefficient in the residuals.
- U1 [ tseries ] - Low-frequency regression residuals.
- U2 [ tseries ] - Higher-frequency regression residuals.

## Options

- 'constant=' [ true | false ] - Include a constant term in the regression.
- 'log=' [ true | false ] - Logarithmise the data before distribution, de-logarithmise afterwards.
- 'ngrid=' [ numeric | 200 ] - Number of grid search points for finding autocorrelation coefficient for higher-frequency residuals.
- 'rho=' [ 'estimate' | 'positive' | 'negative' | numeric ] - How to determine the autocorrelation coefficient for higher-frequency residuals.
- 'timeTrend=' [ true | false ] - Include a time trend in the regression.

## Description

Chow,G.C., and A.Lin (1971). Best Linear Unbiased Interpolation, Distribution and Extrapolation of Time Series by Related Times Series. Review of Economics and Statistics, 53, pp. 372–75.

See also Appendix 2 in Robertson, J.C., and E.W.Tallman (1999). Vector Autoregressions: Forecasting and Reality. FRB Atlanta Economic Review, 1st Quarter 1999, pp.4–17.

## Example

---

## ■ convert

Convert tseries object to a different frequency

## Syntax

```
Y = convert(X,NEWFREQ)
Y = convert(X,NEWFREQ,RANGE,...)
```

## Input arguments

- X [ tseries ] - Input tseries object that will be converted to a new frequency, freq, aggregating or interpolating the data.

## Time series objects and functions: convert

- NEWFREQ [ numeric | char ] - New frequency to which the input data will be converted: 1 or 'A' for annual, 2 or 'H' for half-yearly, 4 or 'Q' for quarterly, 6 or 'B' for bi-monthly, and 12 or 'M' for monthly.
- RANGE [ numeric ] - Date range on which the input data will be converted.

### Output arguments

- Y [ tseries ] - Output tseries created by converting X to the new frequency.

### Options

- 'ignoreNaN=' [ true | false ] - Exclude NaNs from aggregation.
- 'missing=' [ numeric | NaN | 'last' ] - Replace missing observations with this value.

### Options for high- to low-frequency conversion (aggregation)

- 'method=' [ function\_handle | 'first' | 'last' | @mean ] - Method that will be used to aggregate the high frequency data.
- 'select=' [ numeric | Inf ] - Select only these high-frequency observations within each low-frequency period; Inf means all observations will be used.

### Options for low- to high-frequency conversion (interpolation)

- 'method=' [ char | 'cubic' | 'quadsum' | 'quadavg' ] - Interpolation method; any option available in the built-in interp1 function can be used.
- 'position=' [ 'centre' | 'start' | 'end' ] - Position of the low-frequency date grid.

### Description

The function handle that you pass in through the 'method' option when you aggregate the data (convert higher frequency to lower frequency) should behave like the built-in functions mean, sum etc. In other words, it is expected to accept two input arguments:

- the data to be aggregated,
- the dimension along which the aggregation is calculated.

The function will be called with the second input argument set to 1, as the data are processed en block columnwise. If this call fails, convert will attempt to call the function with just one input argument, the data, but this is not a safe option under some circumstances since dimension mismatch may occur.

## Example

---

## ■ cumsumk

Cumulative sum with a k-period leap

### Syntax

```
Y = cumsumk(X,K,RHO,RANGE)
Y = cumsumk(X,K,RHO)
Y = cumsumk(X,K)
Y = cumsumk(X)
```

### Input arguments

- X [ tseries ] - Input data.
- K [ numeric ] - Number of periods that will be leapt the cumulative sum will be taken; if not specified, K is chosen to match the frequency of the input data (e.g. K = -4 for quarterly data), or K = -1 for indeterminate frequency.
- RHO [ numeric ] - Autoregressive coefficient; if not specified, RHO = 1.
- RANGE [ numeric ] - Range on which the cumulative sum will be computed and the output series returned.

### Output arguments

- Y [ tseries ] - Output data constructed as described below.

### Options

- 'log=' [ true | false ] - Logarithmise the input data before, and de-logarithmise the output data back after, running  $\times 12$ .

**Description**

If  $K < 0$ , the first  $K$  observations in the output series  $Y$  are copied from  $X$ , and the new observations are given recursively by

$$Y\{t\} = \text{RHO} * Y\{t-K\} + X\{t\}.$$

If  $K > 0$ , the last  $K$  observations in the output series  $Y$  are copied from  $X$ , and the new observations are given recursively by

$$Y\{t\} = \text{RHO} * Y\{t+K\} + X\{t\},$$

going backwards in time.

If  $K == 0$ , the input data are returned.

**Example**

Construct random data with seasonal pattern, and run X12 to seasonally adjust these series.

```
x = tseries(qq(1990,1):qq(2020,4),@randn);
x1 = cumsumk(x,-4,1);
x2 = cumsumk(x,-4,0.7);
x1sa = x12(x1);
x2sa = x12(x2);
```

The new series  $x1$  will be a unit-root process while  $x2$  will be stationary. Note that the command on the second line could be replaced with  $x1 = \text{cumsumk}(x)$ .

**■ DAILY**

Calendar view of a daily tseries object

**Syntax**

```
daily(X)
```



### Input arguments

- `X [ tseries ]` - Tseries object with indeterminate frequency whose date ticks will be interpreted as Matlab serial date numbers.

### Description

### Example

---

## ■ destdise

Destandardise tseries object by applying specified standard deviation and mean to it

### Syntax

```
X = destdise(X,XMEAN,XSTD)
```

### Input arguments

- `X [ tseries ]` - Input tseries object.
- `XMEAN [ numeric ]` - Mean that will be added the data.
- `XSTD [ numeric ]` - Standard deviation that will be added to the data.

### Output arguments

- `X [ tseries ]` - Destandardised output data.

### Description

### Example

---

## ■ detrend

Remove a linear time trend

### Syntax

```
x = detrend(x,range)
```

### Input arguments

- x [ tseries ] - Input time series.
- range [ tseries ] - Range for which the trend will be computed.

### Output arguments

- x [ tseries ] - Output time series with a trend removed.

### Options

See [tseries/trend](#) P303 for options available.

### Description

### Example

```
}
```

---

## ■ diff

First difference

### Syntax

```
Y = diff(X)  
Y = diff(X,K)
```

### Input arguments

- `X [ tseries ]` - Input tseries object.
- `K [ numeric ]` - Number of periods over which the first difference will be computed;  $Y = X - X\{K\}$ . Note that `K` must be a negative number for the usual backward differencing. If not entered, the default `-1` will be used.

### Output arguments

- `Y [ tseries ]` - First difference of the input data.

### Description

### Example

---

## ■ double

Return tseries observations as double-precision numeric array

### Syntax

```
y = double(x)
```

### Input arguments

- `x [ tseries ]` - Tseries object whose observations will be returned as double-precision numeric array.

### Output arguments

- `y [ numeric ]` - Double-precision numeric array with the input tseries observations in columns.

## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ doubledata

Convert tseries observations to double precision

## Syntax

```
x = doubledata(x)
```

## Input arguments

- `x [ tseries ]` - Tseries object whose observations will be converted to double precision.

## Output arguments

- `y [ numeric ]` - Tseries object with double-precision observations.

## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ empty

Empty tseries object preserving its size in 2nd and higher dimensions

## Syntax

```
x = empty(x)
```

### Input arguments

- `x [ tseries ]` - Tseries object that will be emptied.

### Output arguments

- `x [ tseries ]` - Empty tseries object with the 2nd and higher dimensions the same size as the input tseries object, and comments preserved.

### Description

### Example

---

## ■ enddate

Date of the last available observation in a tseries object

### Syntax

```
d = enddate(x)
```

### Input arguments

- `x [ tseries ]` - Tseries object.

### Output arguments

- `d [ numeric ]` - IRIS serial date number representing the date of the last observation available in the input tseries.

### Description

The startdate function is equivalent to calling

```
get(x,'endDate')
```

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ errorbar

Line plot with error bars

### Syntax

```
[LL,EE,RANGE] = errorbar(X,B,...)
[LL,EE,RANGE] = errorbar(RANGE,X,B,...)
[LL,EE,RANGE] = errorbar(AA,RANGE,X,B,...)
[LL,EE,RANGE] = errorbar(X,LO,HI,...)
[LL,EE,RANGE] = errorbar(RANGE,X,LO,HI,...)
[LL,EE,RANGE] = errorbar(AA,RANGE,X,LO,HI,...)
```

### Input arguments

- AA [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- RANGE [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [ tseries ] - Tseries object whose data will be plotted as a line graph.
- B [ tseries ] - Width of the bands that will be plotted around the lines.
- LO [ tseries ] - Width of the band below the line.
- HI [ tseries ] - Width of the band above the line.

### Output arguments

- LL [ numeric ] - Handles to lines plotted.
- EE [ numeric ] - Handles to error bars plotted.
- RANGE [ numeric ] - Actually plotted date range.

## Options

- 'dateformat=' [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- 'datetick=' [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.
- 'relative=' [ true | false ] - If true, the data for the lower and upper bounds are relative to the centre, i.e. the bounds will be added to the centre (in this case, LOW must be negative numbers and HIGH must be positive numbers). If false, the bounds are absolute data (in this case LOW must be lower than X, and HIGH must be higher than X).

See help on built-in plot function for other options available.

---

## ■ ews

### Exponential smoothing

#### Syntax

```
X = expsmooth(X,BETA,...)
```

#### Input arguments

- X [ tseries ] - Input time series.
- BETA [ numeric ] - Exponential factor.

#### Output arguments

- X [ tseries ] - Exponentially smoothed series.

## Options

- 'init=' [ numeric | NaN ] - Add this value before the first observation to initialise the smoothing.
- 'log=' [ true | false ] - Logarithmise the data before filtering, de-logarithmise afterwards.

## Description

## Examples

---

### ■ `fft`

Discrete Fourier transform of tseries object

## Syntax

```
[y,range,freq,per] = fft(x)
[y,range,freq,per] = fft(x,range,...)
```

## Input arguments

- `x [ tseries ]` - Input tseries object that will be transformed.
- `range [ numeric | Inf ]` - Date range.

## Output arguments

- `y [ numeric ]` - Fourier transform with data organised in columns.
- `range [ numeric ]` - Actually used date range.
- `freq [ numeric ]` - Frequencies corresponding to FFT vector elements.
- `per [ numeric ]` - Periodicities corresponding to FFT vector elements.

## Options

- `'full=' [ true | false ]` - Return Fourier transform on the whole interval  $[0,2\pi]$ ; if false only the interval  $[0,\pi]$  is returned.



## Description

## Example

```
}
```

---

## ■ freq

Frequency of a tseries object

## Syntax

```
f = freq(x)
```

## Input arguments

- `x [ tseries ]` - Tseries object.

## Output arguments

- `f [ 0 | 1 | 2 | 4 | 6 | 12 ]` - Frequency of observations in the input tseries object (`f` is the number of periods within a year).

## Description

The `freq` function is equivalent to calling

```
get(x, 'freq')
```

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ get

Query tseries object property

### Syntax

```
value = get(x,query)
[value,value,...] = get(x,query,query,...)
```

### Input arguments

- x [ model ] - Tseries object.
- query [ char ] - Name of the queried property.

### Output arguments

- value [ ... ] - Value of the queried property.

### Valid queries on tseries objects

- 'end=' Returns [ numeric ] the date of the last observation.
- 'freq=' Returns [ numeric ] the frequency (periodicity) of the time series.
- 'nanEnd=' Returns [ numeric ] the last date at which observations are available in all columns; for scalar tseries, this query always returns the same as 'end'.
- 'nanRange=' Returns [ numeric ] the date range from 'nanstart' to 'nanend'; for scalar time series, this query always returns the same as 'range'.
- 'nanStart=' Returns [ numeric ] the first date at which observations are available in all columns; for scalar tseries, this query always returns the same as 'start'.
- 'range=' Returns [ numeric ] the date range from the first observation to the last observation.
- 'start=' Returns [ numeric ] the date of the first observation.

### Description

---

## ■ hpdi

Highest probability density interval

### Syntax

```
int = hpdi(x,prob)
```

### Input arguments

- `x [ tseries ]` - Input data with random draws in each period.
- `prob [ numeric ]` - Percent coverage of the computed interval, between 0 and 100.

### Output arguments

- `int [ tseries ]` - Output tseries object with two columns, i.e. lower bounds and upper bounds for each period.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ hpf

Hodrick-Prescott filter with tunes

### Syntax

```
[T,C] = hpf(X)  
[T,C] = hpf(X,RANGE,...)
```

**Syntax with output arguments swapped**

```
[C,T] = hpf2(X)
[C,T] = hpf2(X,RANGE,...)
```

**Input arguments**

- `X [ tseries ]` - Input tseries object that will be filtered.
- `RANGE [ numeric ]` - Date range on which the input data will be filtered; RANGE can be `Inf`, `[startdata,Inf]`, or `[-Inf,enddate]`; if not specified, `Inf` (i.e. the entire available range of the input series) is used.

**Output arguments**

- `T [ tseries ]` - Lower-frequency (trend) component.
- `C [ tseries ]` - Higher-frequency (cyclical) component.

**Options**

- `'lambda=' [ numeric | 100 freq^2 ]` - Smoothing parameter; needs to be specified for tseries objects with indeterminate frequency.
- `'level=' [ tseries ]` - Hard tunes on the level of the trend.
- `'change=' [ tseries ]` - Hard tunes on the change of the trend.
- `'log=' [ true | false ]` - Logarithmise the data before filtering, de-logarithmise afterwards.

**Description**

*Design of the optimisation problem*      The hpf function solves the following constrained optimisation problem:

$$\min_{\bar{y}_t} \lambda \sum_t (\Delta \bar{y}_t - \Delta \tilde{y}_t)^2 + \sum_{t \in \Omega} (\bar{y}_t - y_t)^2,$$

where  $\Delta$  is the first-difference operator,  $\lambda$  is the smoothing parameter,  $\bar{y}_t$  is the computed trend,  $y_t$  are the actual data, and  $\Omega$  is the set of periods in which the observations are available, subject to

- constraints on the level of the trend:

$$\bar{y}_\tau = a_\tau, \quad \tau \in \Sigma_1,$$

where  $a_\tau$  are the user-specified hard tunes, and  $\Sigma_1$  is the set of periods in which the user has imposed constraints on the level of the trend;

- constraints on the change in trend:

$$\Delta \bar{y}_\tau = b_\tau, \quad \tau \in \Sigma_2.$$

where  $b_\tau$  are the user-specified hard tunes and  $\Sigma_2$  is the set of periods in which the user has imposed constraints on the change in the trend.

*Default smoothing parameters* If the user does not specify the smoothing parameter using the 'lambda=' option, a default value is used. The default value is based on the date frequency of the input time series, and is calculated as  $\lambda = 100 \cdot f^2$ , where  $f$  is the frequency (yearly=1, half-yearly=2, quarterly=4, bi-monthly=6, monthly=12). This gives the following default values:

- 100 for yearly time series,
- 400 for half-yearly time series,
- 1600 for quarterly time series,
- 3600 for bi-monthly time series,
- 14400 for monthly time series.

Note that there is no default value for data with indeterminate or daily frequency. For these time series, you must supply the 'lambda=' option.

### Example

---

## ■ hpf2

Swap output arguments of the Hodrick-Prescott filter with tunes

See help on [tseries/hpf](#) P275.

---

## ■ interp

Interpolate missing observations

### Syntax

```
x = interp(x,range,varargin)
```

### Input arguments

- x [ tseries ] - Tseries object.
- range [ tseries ] - Date range on which any missing, i.e. NaN, observations will be interpolated.

### Output arguments

- x [ tseries ] - Tseries object with the missing observations interpolated.

### Options

- 'method=' [ char | 'cubic' ] - Any valid method accepted by the built-in interp1 function.

### Description

### Example

---

## ■ length

Length of tseries object

### Syntax

```
n = length(x)
```

**Input arguments**

- `x [ tseries ]` Tseries object.

**Output arguments**

- `n [ numeric ]` - Number of periods from the first to the last available observation in the input tseries object.

**Description****Example**


---

**■ llf**

Local linear trend (or random-walk-plus-noise) filter with tunes

**Syntax**

```
[T,C] = llf(X)
[T,C] = llf(X,RANGE,...)
```

**Syntax with output arguments swapped**

```
[C,T] = llf2(X)
[C,T] = llf2(X,RANGE,...)
```

**Input arguments**

- `X [ tseries ]` - Input tseries object that will be filtered.
- `RANGE [ numeric ]` - Date range on which the input data will be filtered; `RANGE` can be `Inf`, `[startdata,Inf]`, or `[-Inf,enddate]`; if not specified, `Inf` (i.e. the entire available range of the input series) is used.

### Output arguments

- `T [ tseries ]` - Lower-frequency (trend) component.
- `C [ tseries ]` - Higher-frequency (cyclical) component.

### Options

- `'drift=' [ numeric | tseries | 0 ]` - Deterministic drift in the trend.
- `'lambda=' [ numeric | 10 freq ]` - Smoothing parameter; needs to be specified for tseries objects with indeterminate frequency.
- `'level=' [ tseries ]` - Hard tunes on the level of the trend.
- `'change=' [ tseries ]` - Hard tunes on the change of the trend.
- `'log=' [ true | false ]` - Logarithmise the data before filtering, de-logarithmise afterwards.

### Description

*Design of the optimisation problem* The llf function solves the following constrained optimisation problem:

$$\min_{\bar{y}_t} \lambda \sum_t (\Delta \bar{y}_t - \delta_t)^2 + \sum_{t \in \Omega} (\bar{y}_t - y_t)^2,$$

where  $\Delta$  is the first-difference operator,  $\lambda$  is the smoothing parameter,  $\delta_t$  is a user-specified drift,  $\bar{y}_t$  is the computed trend,  $y_t$  are the actual data, and  $\Omega$  is the set of periods in which the observations are available, subject to

- constraints on the level of the trend:

$$\bar{y}_\tau = a_\tau, \quad \tau \in \Sigma_1,$$

where  $a_\tau$  are the user-specified hard tunes, and  $\Sigma_1$  is the set of periods in which the user has imposed constraints on the level of the trend;

- constraints on the change in trend:

$$\Delta \bar{y}_\tau = b_\tau, \quad \tau \in \Sigma_2.$$

where  $b_\tau$  are the user-specified hard tunes and  $\Sigma_2$  is the set of periods in which the user has imposed constraints on the change in the trend.



*Default smoothing parameters* If you do not specify the smoothing parameter using the 'lambda=' option, a default value is used. The default value is based on the date frequency of the input time series, and is calculated as  $\lambda = 10 \cdot f$ , where  $f$  is the frequency (yearly=1, half-yearly=2, quarterly=4, bi-monthly=6, monthly=12). This gives the following default values:

- 10 for yearly time series,
- 20 for half-yearly time series,
- 40 for quarterly time series,
- 60 for bi-monthly time series,
- 120 for monthly time series.

Note that there is no default value for data with indeterminate or daily frequency. For these time series, you must always supply the 'lambda=' option.

### Example

---

## ■ llf2

Swap output arguments of the local linear trend filter with tunes

See help on [tseries/llf](#) P279.

---

## ■ moving

Apply function to moving window of observations

### Syntax

```
X = moving(X)
X = moving(X,RANGE,...)
```

### Input arguments

- `X [ tseries ]` - Tseries object on whose observations the function will be applied.
- `RANGE [ numeric | Inf ]` - Range on which the moving function will be applied; `Inf` means the entire range on which the time series is defined.

### Output arguments

- `X [ tseries ]` - Output time series.

### Options

- `'function=' [ function_handle | @mean ]` - Function to be applied to a moving window of observations.
- `'window=' [ numeric | Inf ]` - The window of observations where 0 means the current date, -1 means one period lag, etc. `Inf` means that the last `n` observations (including the current one) are used, where `n` is the frequency of the input data.

### Description

### Example

---

## ■ ndims

Number of dimensions in tseries object data

### Syntax

`N = ndims(X)`

### Input arguments

- `X [ tseries ]` - Input tseries object.

### Output arguments

- N [ numeric ] - Number of dimensions in the input object.

### Description

### Example

---

## ■ normalise

Normalise data to particular date

### Syntax

```
x = normalise(x)
x = normalise(x,normdate,...)
```

### Input arguments

- x [ tseries ] - Input tseries object that will be normalised.
- normdate [ numeric | 'start' | 'end' | 'nanstart' | 'nanend' ] - Date relative to which the input data will be normalised; if not specified, 'nanstart' (the first date for which all columns have an observation) will be used.

### Output arguments

- x [ tseries ] - Normalised tseries object.

### Options

- 'mode=' [ 'add' | 'mult' ] - Additive or multiplicative normalisation. Description =====

### Example

---

## ■ pct

Percent rate of change

### Syntax

```
x = pct(x)
x = pct(x,k)
```

### Input arguments

- x [ tseries ] - Input tseries object.
- k [ numeric ] - Time shift over which the rate of change will be computed, i.e. between time t and t+k; if not specified k will be set to -1.

### Output arguments

- x [ tseries ] - Percentage rate of change in the input data.

### Description

### Example

---

## ■ permute

Permute dimensions of a tseries object

### Syntax

```
X = permute(X,ORDER)
```

### Input arguments

- X [ tseries ] - Tseries object whose dimensions, except the first (time) dimension, will be rearranged in the order specified by the vector order.

- ORDER [ numeric ] - New order of dimensions; because the time dimension cannot be permuted, order(1) must be always 1.

#### Output arguments

- x [ tseries ] - Output tseries object with its dimensions permuted.

#### Description

#### Example

---

## ■ plot

Line graph for tseries objects

#### Syntax

```
[h,range] = plot(x,...)
[h,range] = plot(range,x,...)
[h,range] = plot(a,range,x,...)
```

#### Input arguments

- a [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- range [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- x [ tseries ] - Input tseries object whose columns will be plotted as a line graph.

#### Output arguments

- h [ numeric ] - Handles to the lines plotted.
- range [ numeric ] - Actually plotted date range.

## Options

- 'dateFormat=' [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- 'datePosition=' [ 'centre' | 'end' | 'start' ] - Position of each date point within a given period span.
- 'datetick=' [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.

See help on built-in plot function for other options available.

## Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ plotcmp

Comparison graph for two time series

### Syntax

```
[AA,LL,RR] = plotcmp(X,...)
[AA,LL,RR] = plotcmp(RANGE,X,...)
```

### Input arguments

- RANGE [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- X [ tseries ] - Tseries object with two columns; the difference between the second and the first column will be displayed as an area or bar graph.

### Output arguments

- AA [ numeric ] - Handles to the LHS and RHS axes.
- LL [ numeric ] - Handles to the two original lines.
- RR [ numeric ] - Handles to the area or bar difference graph.

### Options

- 'diffColor=' [ numeric | [1,0.75,0.75] ] - Color that will be used to plot the area or bar difference graph.
- 'diffPlotFunc=' [ @area | @bar ] - Function that will be used to plot the difference data on the RHS.

See help on [tseries/plotyy](#) P288 for other options available.

### Description

### Example

---

## ■ plotpred

Plot Kalman filter predictions

### Syntax

```
[H1,H2] = plotpred([X,Y],...)  
[H1,H2] = plotpred(AX,[X,Y],...)  
[H1,H2] = plotpred(AX,RANGE,[X,Y],...)
```

### Input arguments

- X [ tseries ] - Input data with time series observations.
- Y [ tseries ] - Input data with predictions calculated in a Kalman filter run with an 'ahead=' option.

- AX [ numeric ] - Handle to axes object in which the data will be plotted.
- RANGE [ numeric | Inf ] - Date range on which the input data will be plotted.

### Output arguments

- H1 [ numeric ] - Handle to a line object showing the time series observations.
- H2 [ numeric ] - Handle to line objects showing the Kalman filter predictions.

### Options

See help on [plot](#) P285 and on the built-in function plot for options available.

### Description

### Example

---

## ■ plotyy

Line plot function with LHS and RHS axes for time series

### Syntax

```
[ax,lhs,rhs,range] = plotyy(x,y,...)
[ax,lhs,rhs,range] = plotyy(range,x,y,...)
```

### Input arguments

- range [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- x [ tseries ] - Input tseries object whose columns will be plotted and labelled on the LHS.
- y [ tseries ] - Input tseries object whose columns will be plotted and labelled on the RHS.



### Output arguments

- `ax` [ numeric ] - Handles to the LHS and RHS axes.
- `lhs` [ numeric ] - Handles to lines belonging to the LHS axis.
- `rhs` [ numeric ] - Handles to lines belonging to the RHS axis.
- `range` [ numeric ] - Actually plotted date range.

### Options

- `'conincident='` [ true | false ] - Make the LHS and RHS y-axis grids coincident.
- `'dateFormat='` [ char | `irisget('plotdateformat')` ] - Date format for the tick marks on the x-axis.
- `'dateTick='` [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.
- `'freqLetters='` [ char | 'YHQBm' ] - Five letters to represent the five date frequencies (yearly, half-yearly, quarterly, bi-monthly, and monthly).
- `'lhsPlotFunc='` [ @area | @bar | @plot | @stem ] - Function that will be used to plot the LHS data.
- `'lhsTight='` [ true | false ] - Make the LHS y-axis tight.
- `'rhsPlotFunc='` [ @area | @bar | @plot | @stem ] - Function that will be used to plot the RHS data.
- `'rhsTight='` [ true | false ] - Make the RHS y-axis tight.

### Description

### Example

---

## ■ `redate`

Change time dimension of a `tseries` object

### Syntax

```
x = redate(x,oldDate,newDate)
```

### Input arguments

- x [ tseries ] - Input tseries object.
- oldDate [ numeric ] - Base date that will be converted to a new date.
- newDate [ numeric ] - A new date to which the base date oldDate will be changed; newDate can need not be the same frequency as oldDate.

### Output arguments

- x [ tseries ] - Output tseries object with identical data as the input tseries object, but with its time dimension changed.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ regress

Ordinary or weighted least-square regression

### Syntax

```
[b,bstd,e,estd,yfit,range,bcov] = regress(y,x)  
[b,bstd,e,estd,yfit,range,bcov] = regress(y,x,range,...)
```

### Input arguments

- y [ tseries ] - Tseries object with independent (LHS) variables.
- x [ tseries ] - Tseries object with regressors (RHS) variables.

- `range [ numeric ]` - Date range on which the regression will be run; if not specified, the entire range available will be used.

### Output arguments

- `b [ numeric ]` - Vector of estimated regression coefficients.
- `bstd [ numeric ]` - Vector of std errors of the estimates.
- `e [ tseries ]` - Tseries object with the regression residuals.
- `estd [ numeric ]` - Estimate of the std deviation of the regression residuals.
- `yfit [ tseries ]` - Tseries object with fitted LHS variables.
- `range [ numeric ]` - The actually used date range.
- `bcov [ numeric ]` - Covariance matrix of the coefficient estimates.

### Options

- `'constant=' [ true | false ]` - Include a constant vector in the regression; if true the constant will be placed last in the matrix of regressors.
- `'weighting=' [ tseries | empty ]` - Tseries object with weights on the observations in the individual periods.

### Description

This function calls the built-in `lscov` function.

### Example

```
}
```

---

## ■ reshape

Reshape size of time series in 2nd and higher dimensions

### Syntax

```
x = reshape(x, newsize)
```

### Input arguments

- `x [ tseries ]` - Tseries object whose data will be reshaped in 2nd and/or higher dimensions.
- `newsize [ numeric ]` - New size of the tseries object data; the first dimension (time) must be preserved.

### Output arguments

- `x [ tseries ]` - Reshaped tseries object.

### Description

### Example

```
}
```

---

## ■ `resize`

Clip tseries object down to specified date range

### Syntax

```
x = resize(x, range)
```

### Input arguments

- `x [ tseries ]` - Input tseries object whose date range will be clipped down.
- `range [ numeric ]` - New date range to which the input tseries object will be clipped down.

### Output arguments

- `x [ tseries ]` - Output tseries object with its date range clipped down to range.

### Description

### Example

---

## ■ round

Round tseries data to specified number of decimals

### Syntax

```
x = round(x)
x = round(x,dec)
```

### Input arguments

- `x [ tseries ]` - Tseries object whose data will be rounded.
- `dec [ numeric ]` - Number of decimals to which the tseries data will be rounded; if not specified, the data are rounded to nearest integer.

### Output arguments

- `x [ tseries ]` - Rounded tseries object.

### Description

The number of decimals, to which the tseries data will be rounded, can be positive, zero, or negative.

### Example

-IRIS Toolbox. -Copyright (c) Jaromir Benes.

---

## ■ scatter

Scatter graph for tseries objects

### Syntax

```
[h,range] = scatter([x,y],...)  
[h,range] = scatter(range,[x,y],...)  
[h,range] = scatter(a,range,[x,y],...)
```

### Input arguments

- a [ numeric ] - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- range [ numeric ] - Date range; if not specified the entire range of the input tseries object will be plotted.
- x, y [ tseries ] - Two scalar tseries objects plotted on the x-axis and the y-axis, respectively.

### Output arguments

- h [ numeric ] - Handles to the lines plotted.
- range [ numeric ] - Actually plotted date range.

### Options

- 'dateformat=' [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- 'datetick=' [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.

See help on built-in plot function for other options available.

## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ single

Return tseries observations as single-precision numeric array

## Syntax

```
y = single(x)
```

## Input arguments

- `x [ tseries ]` - Tseries object whose observations will be returned as single-precision numeric array.

## Output arguments

- `y [ numeric ]` - Single-precision numeric array with the input tseries observations in columns.

## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ singledata

Convert tseries observations to single precision

### Syntax

```
x = singledata(x)
```

### Input arguments

- `x [ tseries ]` - Tseries object whose observations will be converted to single precision.

### Output arguments

- `y [ numeric ]` - Tseries object with single-precision observations.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ size

Size of tseries object data

### Syntax

```
S = size(X)  
[S1,S2,...,Sn] = size(X)
```

### Input arguments

- `X [ tseries ]` - Tseries object whose size will be returned.

### Output arguments

- `S [ numeric ]` - Vector of sizes of the tseries object data in each dimension,  $S = [S1, S2, \dots, Sn]$ .
- `S1, S2, \dots, Sn [ numeric ]` - Sizes of the tseries object data in each dimension.



## Description

## Example

---

### ■ sort

Sort tseries columns by specified criterion

## Syntax

```
[Y,INDEX] = sort(X,CRIT)
```

## Input arguments

- X [ tseries ] - Input tseries object whose columns will be sorted in order determined by the criterion crit.
- CRIT [ 'sumsq' | 'sumabs' | 'max' | 'maxabs' | 'min' | 'minabs' ] - Criterion used to sort the input tseries object columns.

## Output arguments

- Y [ tseries ] - Output tseries object with columns sorted in order determined by the input criterion, CRIT.
- INDEX [ numeric ] - Vector of indices,  $y = x\{:, \text{index}\}$ .

## Description

## Example

---

### ■ spy

Visualise tseries observations that pass a test

**Syntax**

```
[AA,LL] = spy(X,...)
[AA,LL] = spy(RANGE,X,...)
```

**Input arguments**

- X [ tseries ] - Input tseries object whose non-NaN observations will be plotted as markers.
- RANGE [ tseries ] - Date range on which the tseries observations will be visualised; if not specified the entire available range will be used.

**Output arguments**

- AA [ tseries ] - Handle to the axes created.
- LL [ tseries ] - Handle to the marks plotted.

**Options**

- 'dateformat=' [ char | irisget('plotdateformat') ] - Date format for the tick marks on the x-axis.
- 'datetick=' [ numeric | Inf ] - Vector of dates locating tick marks on the x-axis; Inf means they will be created automatically.
- 'names=' [ cellstr ] - Names that will be used to annotate individual columns of the input tseries object.
- 'test=' [ function\_handle | @(x)~isnan(x) ] - Test applied to each observations; only the values returning a true will be displayed.

**Description****Example**

---

**■ startdate**

Date of the first available observation in a tseries object

### Syntax

```
d = startdate(x)
```

### Input arguments

- `x [ tseries ]` - Tseries object.

### Output arguments

- `d [ numeric ]` - IRIS serial date number representing the date of the first observation available in the input tseries.

### Description

The startdate function is equivalent to calling

```
get(x,'startDate')
```

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ stdise

Standardise tseries data by subtracting mean and dividing by std deviation

### Syntax

```
[X,M,S] = stdise(X)  
[X,M,S] = stdise(X,FLAG)
```

### Input arguments

- `X [ tseries ]` - Input tseries object whose data will be normalised.
- `FLAG [ 0 | 1 ]` - `flag==0` normalises by  $N-1$ , `flag==1` normalises by  $N$ , where  $N$  is the sample length.

### Output arguments

- `X [ tseries ]` - Output tseries object with standardised data.
- `XMEAN [ numeric ]` - Estimated mean subtracted from the input tseries observations.
- `XSTD [ numeric ]` - Estimated std deviation by which the input tseries observations have been divided.

### Description

### Example

---

## ■ stem

Plot tseries as discrete sequence data

### Syntax

```
[h,range] = stem(x,...)
[h,range] = stem(range,x,...)
[h,range] = stem(a,range,x,...)
```

### Input arguments

- `a [ numeric ]` - Handle to axes in which the graph will be plotted; if not specified, the current axes will be used.
- `range [ numeric ]` - Date range; if not specified the entire range of the input tseries object will be plotted.
- `x [ tseries ]` - Input tseries object whose columns will be plotted as a stem graph.

### Output arguments

- `h` [ numeric ] - Vector of handles to the stems plotted.
- `range` [ numeric ] - Actually plotted date range.

### Options

- `'dateformat='` [ char | `irisget('plotdateformat')` ] - Date format for the tick marks on the x-axis.
- `'datetick='` [ numeric | `Inf` ] - Vector of dates locating tick marks on the x-axis; `Inf` means they will be created automatically.

See help on built-in `stem` function for other options available.

### Description

### Example

---

## ■ subsasgn

Subscripted assignment for tseries objects

### Syntax

```
x(dates) = values;  
x(dates,i,j,k,...) = values;
```

### Input arguments

- `x` [ tseries ] - Tseries object that will be assigned new observations.
- `dates` [ numeric ] - Dates for which the new observations will be assigned.
- `i, j, k, ...` [ numeric ] - References to 2nd and higher dimensions of the tseries object.
- `values` [ numeric ] - New observations that will assigned at specified dates.

### Output arguments

- `x [ tseries ]` - Tseries object with newly assigned observations.

### Description

### Example

-IRIS Toolbox. -Copyright 2007–2012 Jaromir Benes.

---

## ■ subsref

Subscripted reference function for tseries objects

### Syntax returning numeric array

```
... = x(dates)
... = x(dates,...)
```

### Syntax returning tseries object

```
... = x{dates}
... = x{dates,...}
```

### Input arguments

- `x [ tseries ]` - Tseries object.
- `dates [ numeric ]` - Dates for which the time series observations will be returned, either as a numeric array or as another tseries object.

### Description

### Example

---

## ■ trend

Estimate a time trend

### Syntax

```
x = trend(x,range)
```

### Input arguments

- x [ tseries ] - Input time series.
- range [ tseries ] - Range for which the trend will be computed.

### Output arguments

- x [ tseries ] - Output trend time series.

### Options

- 'break=' [ numeric | empty ] - Vector of breaking points at which the trend may change its slope.
- 'connect=' [ true | false ] - Works only with 'diff=' true. Connect the first and last observations; otherwise the filtered series is centred on zero.
- 'diff=' [ true | false ] - Estimate the trend on differenced data.
- 'log=' [ true | false ] - Logarithmise the input data, de-logarithmise the output data.
- 'season=' [ true | false | 2 | 4 | 6 | 12 ] - Include deterministic seasonal factors in the trend.

### Description

### Example

---

## ■ tseries

Create new time series (tseries) object

### Syntax

```
X = tseries()  
X = tseries(DATES,VALUES)  
X = tseries(DATES,VALUES,COMMENTS)
```

### Input arguments

- DATES [ numeric ] - Dates for which observations will be supplied; dates do not have to be sorted in ascending order. If dates is scalar and values have multiple rows, then the date in dates is interpreted as a startdate for the time series.
- VALUES [ numeric | function\_handle ] - Numerical values (observations) arranged columnwise, or a function that will be used to create an N-by-1 array of values, where N is the number of dates.
- COMMENTS [ char | cellstr ] - Comment or comments attached to each column of observations.

### Output arguments

- X [ tseries ] - New tseries object.

### Description

### Example

---

## ■ windex

Simple weighted or Divisia index

### Syntax

```
y = windex(x,w,range)
```



### Input arguments

- `x [ tseries ]` - Input times series.
- `w [ tseries | numeric ]` - Fixed or time-varying weights on the input time series.
- `range [ numeric ]` - Range on which the Divisia index is computed.

### Output arguments

- `y [ tseries ]` - Weighted index based on `x`.

### Options

- `'method=' [ 'divisia' | 'simple' ]` - Weighting method.
- `'log=' [ true | false ]` - Logarithmise the input data before computing the index, delogarithmise the output data.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ wmean

Weighted average of time series observations

### Syntax

`Y = wmean(X,RANGE,BETA)`

### Input arguments

- `X [ tseries ]` - Input tseries object whose data will be averaged column by column.
- `RANGE [ numeric ]` - Date range on which the weighted average will be computed.

- BETA [ numeric ] - Discount factor; the last observation gets a weight of 1, the N-minus-1st observation gets a weight of BETA, the N-minus-2nd gets a weight of BETA^2, and so on.

### Output arguments

- Y [ numeric ] - Array with weighted average of individual columns; the sizes of Y are identical to those of the input tseries object in 2nd and higher dimensions.

### Description

### Example

---

## ■ x12

Access to X12 seasonal adjustment program

### Syntax with a single type of output requested

```
[Y,OUTPUTFILE,ERRORFILE,MODEL,X] = x12(X)
[Y,OUTPUTFILE,ERRORFILE,MODEL,X] = x12(X,RANGE,...)
```

### Syntax with multiple types of output requested

```
[Y1,Y2,...,OUTPUTFILE,ERRORFILE,MODEL,X] = x12(X,RANGE,...)
```

See the option 'output=' for the types of output data available from X12.

### Input arguments

- X [ tseries ] - Input data that will seasonally adjusted or filtered by the Census X12 Arima.
- RANGE [ numeric ] - Date range on which the X12 will be run; if not specified or Inf the entire available range will be used.

## Output arguments

- `Y, Y1, Y2, ... [ tseries ]` - Requested output data, by default only one type of output is returned, the seasonally adjusted data; see the option `'output='`.
- `OUTPUTFILE [ cellstr ]` - Contents of the output log files produced by X12; each cell contains the log file for one type of output requested.
- `ERRORFILE [ cellstr ]` - Contents of the error files produced by X12; each cell contains the error file for one type of output requested.
- `MODEL [ struct ]` - Struct array with model specifications and parameter estimates for each of the ARIMA models fitted; `MODEL` matches the size of `X` is 2nd and higher dimensions.
- `X [ tseries ]` - Original input data with forecasts and/or backcasts appended if the options `'forecast='` and/or `'backcast='` are used.

## Options

- `'backcast=' [ numeric | 0 ]` - Run a backcast based on the fitted ARIMA model for this number of periods back to improve on the seasonal adjustment; see help on the x11 specs in the X12-ARIMA manual. The backcast is included in the output argument `X`.
- `'cleanup=' [ true | false ]` - Delete temporary X12 files when done; the temporary files are named `iris_x12a.*`.
- `'log=' [ true | false ]` - Logarithmise the input data before, and de-logarithmise the output data back after, running x12.
- `'forecast=' [ numeric | 0 ]` - Run a forecast based on the fitted ARIMA model for this number of periods ahead to improve on the seasonal adjustment; see help on the x11 specs in the X12-ARIMA manual. The forecast is included in the output argument `X`.
- `'display=' [ true | false ]` - Display X12 output messages in command window; if false the messages will be saved in a TXT file.
- `'dummy=' [ tseries | empty ]` - Dummy variable or variables (in case of a multivariate `tseries` object) used in X12-ARIMA regression; the dummy variables can also include values for forecasts and backcasts if you request them.
- `'dummyType=' [ 'ao' | 'holiday' | 'td' ]` - Type of dummy; see the X12-ARIMA documentation.
- `'mode=' [ 'auto' | 'add' | 'logadd' | 'mult' | 'pseudoadd' | 'sign' ]` - Seasonal adjustment mode (see help on the x11 specs in the X12-ARIMA manual); `'auto'` means that series with only positive or only negative numbers will be adjusted in the `'mult'` (multiplicative) mode, while series with combined positive and negative numbers in the `'add'` (additive) mode.

- 'maxIter=' [ numeric | 1500 ] - Maximum number of iterations for the X12 estimation procedure. See help on the estimation specs in the X12-ARIMA manual.
- 'maxOrder=' [ numeric | [2,1] ] - A 1-by-2 vector with maximum order for the regular ARMA model (can be 1, 2, 3, or 4) and maximum order for the seasonal ARMA model (can be 1 or 2). See help on the automdl specs in the X12-ARIMA manual.
- 'output=' [ char | cellstr | 'SA' ] - List of requested output data; the cellstr or comma-separated list can combine 'IR' for the irregular component, 'SA' for the final seasonally adjusted series, 'SF' for seasonal factors, and 'TC' for the trend-cycle. See also help on the x11 specs in the X12-ARIMA manual.
- 'saveAs=' [ char | empty ] - Name (or a whole path) under which X12-ARIMA output files will be saved.
- 'specFile=' [ char | 'default' ] - Name of the X12-ARIMA spec file; if 'default' the IRIS default spec file will be used, see description.
- 'tdays=' [ true | false ] - Correct for the number of trading days. See help on the x11regression specs in the X12-ARIMA manual.
- 'tolerance=' [ numeric | 1e-5 ] - Convergence tolerance for the X12 estimation procedure. See help on the estimation specs in the X12-ARIMA manual.

## Description

*Spec file* The default X12-ARIMA spec file is +thirdparty/x12/default.spc. You can create your own spec file to include options that are not available through the IRIS interface. You can use the following pre-defined placeholders letting IRIS fill in some of the information needed (check out the default file):

- \$series\_data\$ is replaced with a column vector of input observations;
- \$series\_freq\$ is replaced with a number representing the date frequency: either 4 for quarterly, or 12 for monthly (other frequencies are currently not supported by X12-ARIMA);
- \$series\_startyear\$ is replaced with the start year of the input series;
- \$series\_startper\$ is replaced with the start quarter or month of the input series;
- \$transform\_function\$ is replaced with log or none depending on the mode selected by the user;
- \$forecast\_maxlead\$ is replaced with the requested number of ARIMA forecast periods used to extend the series before seasonal adjustment.

- `$forecast_maxlead$` is replaced with the requested number of ARIMA forecast periods used to extend the series before seasonal adjustment.
- `$tolerance$` is replaced with the requested convergence tolerance in the estimation spec.
- `$maxiter$` is replaced with the requested maximum number of iterations in the estimation spec.
- `$maxorder$` is replaced with two numbers separated by a blank space: maximum order of regular ARIMA, and maximum order of seasonal ARIMA.
- `$x11_mode$` is replaced with the requested mode: 'add' for additive, 'mult' for multiplicative, 'pseudoadd' for pseudo-additive, or 'logadd' for log-additive;
- `$x12_save$` is replaced with the list of the requested output series: 'd10' for seasonals, 'd11' for final seasonally adjusted series, 'd12' for trend-cycle, 'd13' for irregular component.

Two of the placeholders, '`$series_data$`' and '`$x12_output$`', are required; if they are not found in the spec file, IRIS throws an error.

*Estimates of ARIMA model parameters*      The ARIMA model specification, `MODEL`, is a struct with three fields:

- `.spec` - a cell array with the first cell giving the structure of the non-seasonal ARIMA, and the second cell giving the structure of the seasonal ARIMA; both specifications follow the usual Box-Jenkins notation, e.g. `[0 1 1]`.
- `.ar` - a numeric array with the point estimates of the AR coefficients (non-seasonal and seasonal).
- `.ma` - a numeric array with the point estimates of the MA coefficients (non-seasonal and seasonal).

### Example 1

If you wish to run `x12` on the entire range on which the input time series is defined, and do not use any options, you can omit the second input argument (the date range). These following three calls to `x12` do exactly the same:

```
xsa = x12(x);
xsa = x12(x,Inf);
xsa = x12(x,get(x,'range'));
```

## Example 2

If you wish to specify some of the options, you have to enter a date range or use Inf:

```
xsa = x12(x,Inf,'mode=', 'add');
```

---

## ■ yearly

Display tseries object one full year per row

### Syntax

```
yearly(X)
```

### Input arguments

- `X [ tseries ]` - Tseries object that will be displayed one full year of observations per row.

### Description

### Example

## 18 Basic database management

### Loading and saving databases

- `dbload` P319 - Create database by loading CSV file.
- `dbsave` P328 - Save database as CSV file.

### Getting information about databases

- `dbnames` P324 - List of database entries filtered by name or class.
- `dbprintuserdata` P326 - Print names of database tseries along with specified fields of their userdata.
- `dbrange` P327 - Find a range that encompasses the ranges of the listed tseries objects.
- `dbsearchuserdata` P331 - Search database to find tseries by matching the content of their userdata fields.
- `dbuserdatalov` P332 - List of values found in a specified user data field in tseries objects.

### Converting databases

- `array2db` P312 - Convert numeric array to database.
- `db2array` P313 - Convert tseries database entries to numeric array.
- `db2tseries` P313 - Combine tseries database entries in one multivariate tseries object.

### Batch processing

- `dbbatch` P314 - Run a batch job within a database.
- `dbclip` P315 - Clip all database tseries objects down to specified date range.
- `dbcold` P316 - Retrieve the specified column or columns from database entries.
- `dbfun` P318 - Apply function to each database field.
- `dbplot` P325 - Plot from database.
- `dbredate` P328 - Redate all tseries objects in a database.

### Combining databases

- [dbextend](#) P317 - Combine tseries observations from two or more databases.
- [dbmerge](#) P322 - Merge two or more databases.
- [dbminuscontrol](#) P323 - Create simulation-minus-control database.

### Getting on-line help on database functions

```
help dbase  
help dbase/function_name
```

---

## ■ array2db

Convert numeric array to database

### Syntax

```
d = array2db(X,range,name)
```

### Input arguments

- X [ numeric ] - Numeric array with vectors of data in rows.
- range [ numeric ] Date range corresponding to the columns of X.

### Output arguments

- d [ struct ] - Output database.

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.



## ■ db2array

Convert tseries database entries to numeric array

### Syntax

```
[X,included,range] = db2array(d,list,range)
```

### Input arguments

- `d [ struct ]` - Input database with tseries objects that will be converted to a numeric array.
- `list [ char | cellstr ]` - List of tseries names that will be converted to a numeric array.
- `range [ numeric | Inf ]` - Date range.

### Output arguments

- `X [ numeric ]` - Numeric array with observations from individual tseries objects in columns.
- `included [ cellstr ]` - List of tseries names that have been actually found in the database.
- `range [ numeric ]` - Date range actually used.

### Description

### Example

---

## ■ db2tseries

Combine tseries database entries in one multivariate tseries object

### Syntax

```
[X,INCL,RANGE] = db2tseries(D,LIST,RANGE)
```

### Input arguments

- `D [ struct ]` - Input database with tseries objects that will be combined in one multivariate tseries object.
- `LIST [ char | cellstr ]` - List of tseries names that will be combined.
- `RANGE [ numeric | Inf ]` - Date range.

### Output arguments

- `X [ numeric ]` - Combined multivariate tseries object.
- `INCL [ cellstr ]` - List of tseries names that have been actually found in the database.
- `RANGE [ numeric ]` - The date range actually used.

---

## ■ dbbatch

Run a batch job within a database

### Syntax

```
[D,LIST] = dbbatch(D,NAME,COMMAND,...)
```

### Input arguments

- `D [ struct ]` - Input database.
- `NAME [ char ]` - Pattern that will be used to create names for new database entries; the pattern can include `$0`, `$1`, etc to refer to tokens capture in the regular expression specified through the `'nameFilter='` option.
- `COMMAND [ char ]` - Command that will be evaluated on a selection of existing database entries to create new database entries.

### Output arguments

- `D [ struct ]` - Output database.
- `LIST [ cellstr ]` - List of database names that have been processed.

## Options

- 'classFilter=' [ char | Inf ] - From the existing database entries, select only those that are objects of the specified class or classes, and evaluate the COMMAND on these.
- 'fresh=' [ true | false ] - If true, the output database will only contain the newly created entries; if false the output database will also include all the entries from the input database.
- 'nameFilter=' [ char | Inf ] - From the existing database entries, select only those that match this regular expression, and evaluate the COMMAND on these.
- 'nameList=' [ cellstr | Inf ] - Evaluate the COMMAND on this list of existing database entries.
- 'stringList=' [ cellstr | empty ] - Evaluate the COMMAND on this list of strings; the strings do not need to be names existing in the database; this options cannot be comined with 'nameFilter=', 'nameList=', or 'classFilter='.

## Description

### Example

Suppose that in database D you want to seasonally adjust all time series whose names end with \_u, and give these seasonally adjusted series names without the \_u.

```
d = dbbatch(d,'$1','x12(d.$0)','nameFilter','(.*u');
```

or, if you want to make sure only tseries objects will be selected (in case there are database entries ending with a u other than tseries objects)

```
d = dbbatch(d,'$1','x12(d.$0)', ...
    'nameFilter','(.*u','classFilter','tseries');
```

---

## ■ dbclip

Clip all database tseries objects down to specified date range

### Syntax

```
d = dbclip(d,range)
```

### Input arguments

- `d [ struct ]` - Database or nested databases with `tseries` objects.
- `range [ numeric ]` - Range to which all `tseries` objects will be clipped.

### Output arguments

- `d [ struct ]` - Database with `tseries` objects cut down to `range`.

### Description

This functions looks up all `tseries` objects within the database `d`, including `tseries` objects nested in sub-databases, and cuts off any values preceding the start date of `range` or following the end date of `range`. The `tseries` object comments, if any, are preserved in the new database.

### Example

```
d = struct();
d.x = tseries(qq(2005,1):qq(2010,4),@rand);
d.y = tseries(qq(2005,1):qq(2010,4),@rand)

d =
    x: [24x1 tseries]
    y: [24x1 tseries]

dbclip(d,qq(2007,1):qq(2007,4))

ans =
    x: [4x1 tseries]
    y: [4x1 tseries]
```

---

## ■ dbcol

Retrieve the specified column or columns from database entries

### Syntax

```
d = dbcol(d,k)
```

### Input arguments

- `d [ struct ]` - Input database with (possibly) multivariate `tseries` objects and numeric arrays.
- `k [ numeric | logical | 'end' ]` - Column or columns that will be retrieved from each `tseries` object or numeric array and returned in the output database.

### Output arguments

- `d [ struct ]` - Output database with `tseries` objects and numeric arrays reduced to the specified column.

### Description

### Example

---

## ■ dbextend

Combine `tseries` observations from two or more databases

### Syntax

```
D = dbextend(D,D1,D2,...)
```

### Input arguments

- `D [ struct ]` - Primary input database.
- `D1, D2, ... [ struct ]` - Databases whose `tseries` observations will be used to extend or overwrite observations in the `tseries` objects of the same name in the primary database.

### Output arguments

- D [ struct ] - Output database.

### Description

If more than two databases are combined then they are processed one-by-one: the first is combined with the second, then the result is combined with the third, and so on, using the following rules:

- If two non-empty tseries objects with the same frequency are combined, the observations are spliced together. If some of the observations overlap the observations from the second tseries are used.
- If two empty tseries objects are combined the first is used.
- If a non-empty tseries is combined with an empty tseries, the non-empty one is used.
- If two objects are combined of which at least one is a non-tseries object, the second input object is used.

### Example

---

## ■ dbfun

Apply function to each database field

### Syntax

```
[D,FLAG,INVALID] = dbfun(FUNC,D1,...)
[D,FLAG,INVALID] = dbfun(FUNC,D1,D2,...)
```

### Input arguments

- FUNC [ function\_handle | char ] - Function that will be applied to each field.
- D1 [ struct ] - First input database.
- D2 [ struct ] - Second input database (when fun accepts two input arguments).

### Output arguments

- `D` [ struct ] - Output database whose fields will be created by applying `fun` to each field of the input database or databases.
- `FLAG` [ true | false ] - True if no error occurs when evaluating the function.
- `INVALID` [ cellstr ] - List of fields on which the function fails.

### Options

- `'cascade='` [ true | false ] - Cascade through subdatabases applying the function `fun` to their fields, too.
- `'classFilter='` [ cell | cellstr | Inf ] - Apply `fun` only to the fields of selected classes.
- `'fresh='` [ true | false ] - Keep upprocessed fields in the output database.

### Description

### Example

---

## ■ dbload

Create database by loading CSV file

### Syntax

```
D = dbload(FNAME, ...)  
D = dbload(D,FNAME, ...)
```

### Input arguments

- `FNAME` [ char | cellstr ] - Input CSV file name or a cell array of CSV file names that will be combined.
- `D` [ struct ] - An existing database (struct) to which the CSV entries will be added.

## Output arguments

- `D [ struct ]` - Database created from the input CSV file(s).

## Options

- `'case=' [ 'lower' | 'upper' | empty ]` - Change case of variable names.
- `'commentRow=' [ char | cellstr | { 'comment', 'comments' } ]` - Label at the start of row that will be used to create tseries object comments.
- `'dateFormat=' [ char | 'YYYYFP' ]` - Format of dates in first column.
- `'delimiter=' [ char | ',' ]` - Delimiter separating the individual values (cells) in the CSV file; if different from a comma, all occurrences of the delimiter will be replaced with commas — note that this will also affect text in comments.
- `'freq=' [ 0 | 1 | 2 | 4 | 6 | 12 | 365 | 'daily' | empty ]` - Advise frequency of dates; if empty, frequency will be automatically recognised.
- `'freqLetters=' [ char | 'YHQBM' ]` - Letters representing frequency of dates in date column.
- `'leadingRow=' [ char | numeric | empty ]` - String at the beginning (i.e. in the first cell) of the row with variable names; or the line number at which the row with variable names begins.
- `'nameFunc=' [ function_handle | empty ]` - Function used to change or transform the variable names; if empty, the variable names from the CSV are used as they are.
- `'nan=' [ char | NaN ]` - String representing missing observations (case insensitive).
- `'skipRows=' [ char | cellstr | numeric | empty ]` - Skip rows whose first cell matches the string or strings (regular expressions); alternatively, you can specify a vector of row numbers to be skipped.
- `'userdata=' [ char | Inf ]` - Field name under which the database userdata loaded from the CSV file (if they exist) will be stored in the output database; if `'userData'=Inf` the field name will be read from the CSV file (and will be thus identical to the originally saved database).
- `'userdataField=' [ char | '.' ]` - A leading character denoting userdata fields for individual time series. If empty, no userdata fields will be read.

## Description

Use the `'freq='` option whenever there is ambiguity in interpreting the date strings, and IRIS is not able to determine the frequency correctly (see Example 1).



*Structure of CSV database files* The minimalist structure of a CSV database file has a leading row with variables names, a leading column with dates in the basic IRIS format, and individual columns with numeric data:

```
+-----+-----+-----+---
|         |         Y |         P |
+-----+-----+-----+---
| 2010Q1 |         1 |        10 |
+-----+-----+-----+---
| 2010Q2 |         2 |        20 |
+-----+-----+-----+---
|         |         |         |
```

You can add a comment row (must be placed before the data part, and start with a label 'Comment' in the first cell) that will also be read in and assigned as comments to the individual tseries objects created in the output database.

```
+-----+-----+-----+---
|         |         Y |         P |
+-----+-----+-----+---
| Comment | Output | Prices |
+-----+-----+-----+---
| 2010Q1 |         1 |        10 |
+-----+-----+-----+---
| 2010Q2 |         2 |        20 |
+-----+-----+-----+---
|         |         |         |
```

You can use a different label in the first cell to denote a comment row; in that case you need to set the option 'commentRow=' accordingly.

All CSV rows whose names start with a character specified in the option 'userdataField=' (a dot by default) will be added to output tseries objects as fields of their userdata.

```
+-----+-----+-----+---
|         |         Y |         P |
+-----+-----+-----+---
| Comment | Output | Prices |
+-----+-----+-----+---
| .Source | Stat  | IMFIFS |
```

+-----+-----+-----+--			
.Update	17Feb11	01Feb11	
+-----+-----+-----+--			
.Units	Bil USD	2010=1	
+-----+-----+-----+--			
2010Q1	1	10	
+-----+-----+-----+--			
2010Q2	2	20	
+-----+-----+-----+--			

**Example 1**

Typical example of using the 'freq=' option is a quarterly database with dates represented by the corresponding months, such as a sequence 2000-01-01, 2000-04-01, 2000-07-01, 2000-10-01, etc. In this case, you can use the following options:

```
d = dbload('filename.csv','dateFormat','YYYY-MM-01','freq',4);
```

---

## ■ dbmerge

Merge two or more databases

**Syntax**

```
D = dbmerge(D1,D2,...)
```

**Input arguments**

- D1, D2, ... [ struct ] - Input databases whose entries will be combined in the output datase.

**Output arguments**

- D [ struct ] - Output database that combines entries from all input database; if some entries are found in more than one input databases, the last occurence is used.

## Description

### Example

```
d1 = struct('a',1,'b',2);  
d2 = struct('a',10,'c',20);  
d = dbmerge(d1,d2)  
d =  
    a: 10  
    b: 2  
    c: 20
```

---

## ■ dbminuscontrol

Create simulation-minus-control database

### Syntax

```
D = dbminuscontrol(M,D,C)
```

### Input arguments

- M [ model ] - Model object on which the databases D and C are based.
- D [ struct ] - Simulation database.
- C [ struct ] - Control database.

### Output arguments

- D [ struct ] - Simulation-minus-control database, in which all log-variables are d.x/c.x, and all other variables are d.x-c.x.

**Description****Example**

We run a shock simulation in full levels using a steady-state (or balanced-growth-path) database as input, and then compute the deviations from the steady state.

```
d = sstatedb(m,1:40);
... % Set up a shock or shocks here.
s = simulate(m,d,1:40);
s = dbextend(d,s);
s = dbminuscontrol(m,s,d);
```

Note that this is equivalent to running

```
d = zerodb(m,1:40);
... % Set up a shock or shocks here.
s = simulate(m,d,1:40,'deviation',true);
s = dbextend(d,s);
```

---

## ■ dbnames

List of database entries filtered by name or class

**Syntax**

```
LIST = dbnames(D,...)
```

**Input arguments**

- D [ struct ] - Input database.

**Output arguments**

- LIST [ cellstr ] - List of input database entries that pass the name or class test.

## Options

- 'nameFilter=' [ char | Inf ] - Regular expression against which the database entry names will be matched.
- 'classFilter=' [ char | Inf ] - Regular expression against which the database entry class names will be matched.

## Description

## Example

---

# ■ dbplot

Plot from database

## Syntax

```
[FF,AA,PDB] = dbplot(D,LIST,RANGE,...)
[FF,AA,PDB] = dbplot(D,RANGE,LIST,...)
```

## Input arguments

- D [ struct ] - Database with input data.
- LIST [ cellstr ] - List of expressions (or labelled expressions) that will be evaluated and plotted in separate graphs.
- RANGE [ numeric ] - Date range.

## Output arguments

- FF [ numeric ] - Handles to figures created by qplot.
- AA [ cell ] - Handles to axes created by qplot.
- PDB [ struct ] - Database with actually plotted series.

## Options

- 'plotFunc=' [ @bar | @hist | @plot | @stem ] - Plot function used to create the graphs.

See help on [qreport/qplot](#) P372 for other options available.

## Description

The function dbplot opens a new figure window (as many as needed to accommodate all graphs given the option 'subplot='), and creates a graph for each entry in the cell array LIST.

LIST can contain the names of the database time series, expression referring to the database fields evaluating to time series. You can also add labels (that will be displayed as graph titles) enclosed in double quotes and preceding the expressions. If you start the expression with a ^ (hat) symbol, the function specified in the 'transform=' option will not be applied to that expression.

## Example

```
dbplot(d,qq(2010,1):qq(2015,4),{'x','"Series Y" y','^"Series Z"', ...
    'transform=','@(x) 100*(x-1)});
```

---

## ■ dbprintuserdata

Print names of database tseries along with specified fields of their userdata

## Syntax

```
dbprintuserdata(D,FIELD1,FIELD2,...)
```

## Input arguments

- D [ struct ] - Database whose tseries objects will be reported.
- FIELD1, FIELD2, ... [ char ] - Names of the userdata fields whose content (if char) will be reported.

## Description

## Example

---

## ■ dbrange

Find a range that encompasses the ranges of the listed tseries objects

## Syntax

```
[RANGE,FREQ] = dbrange(D,...)
[RANGE,FREQ] = dbrange(D,LIST,...)
```

## Input arguments

- D [ struct ] - Input database.
- LIST [ char | cellstr | Inf ] - List of tseries objects that will be included in the range search; Inf means all tseries objects existing in the input databases will be included.

## Output arguments

- RANGE [ numeric | cell ] - Range that encompasses the observations of the tseries objects in the input database; if tseries objects with different frequencies exist, the ranges are returned in a cell array.
- FREQ [ numeric ] - Date frequencies of the returned ranges.

## Options

- 'startDate=' [ 'maxRange' | 'minRange' ] - 'maxRange' means the range will start at the earliest start date of all tseries included in the search; 'minRange' means the range will start at the latest start date found.
- 'endDate=' [ 'maxRange' | 'minRange' ] - 'maxRange' means the range will end at the latest end date of all tseries included in the search; 'minRange' means the range will end at the earliest end date.

## Description

## Example

---

## ■ dbredate

Redate all tseries objects in a database

## Syntax

```
d = redate(d,oldDate,newDate)
```

## Input arguments

- d [ struct ] - Input database with tseries objects.
- oldDate [ numeric ] - Base date that will be converted to a new date in all tseries objects.
- newDate [ numeric ] - A new date to which the base date oldDate will be changed in all tseries objects; newDate can need not be the same frequency as oldDate.

## Output arguments

- d [ struct ] - Output database where all tseries objects have identical data as in the input database, but with their time dimension changed.

## Description

## Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

## ■ dbsave

Save database as CSV file



## Syntax

```
LIST = dbsave(D,FNAME)
LIST = dbsave(D,FNAME,DATES,...)
```

## Output arguments

- LIST [ cellstr ] - - List of actually saved database entries.

## Input arguments

- D [ struct ] - Database whose tseries and numeric entries will be saved.
- FNAME [ char ] - Filename under which the CSV will be saved, including its extension.
- DATES [ numeric | Inf ] Dates or date range on which the tseries objects will be saved.

## Options

- 'class=' [ true | false ] - Include a row with class and size specifications.
- 'comment=' [ true | false ] - Include a row with comments for tseries objects.
- 'decimal=' [ numeric | empty ] - Number of decimals up to which the data will be saved; if empty the 'format' option is used.
- 'format=' [ char | '%.8e' ] Numeric format that will be used to represent the data, see sprintf for details on formatting, The format must start with a '%', and must not include identifiers specifying order of processing, i.e. the '\$' signs, or left-justify flags, the '-' signs.
- 'freqLetters=' [ char | 'YHQBM' ] - Five letters to represent the five possible date frequencies (annual, semi-annual, quarterly, bimonthly, monthly).
- 'nan=' [ char | 'NaN' ] - String that will be used to represent NaNs.
- 'userData=' [ char | 'userdata' ] - Field name from which any kind of userdata will be read and saved in the CSV file.

## Description

The data saved include also imaginary parts of complex numbers.

*Saving user data with the database* If your database contains field named 'userdata=', this will be saved in the CSV file on a separate row. The 'userdata=' field can be any combination of numeric, char, and cell arrays and 1-by-1 structs.

You can use the 'userdata=' field to describe the database or preserve any sort of metadata. To change the name of the field that is treated as user data, use the 'userData=' option.

### Example 1

Create a simple database with two time series.

```
d = struct();  
d.x = tseries(qq(2010,1):qq(2010,4),@rand);  
d.y = tseries(qq(2010,1):qq(2010,4),@rand);
```

Add your own description of the database, e.g.

```
d.userdata = {'My database',datestr(now())};
```

Save the database as CSV using dbsave,

```
dbsave(d,'mydatabase.csv');
```

When you later load the database,

```
d = dbload('mydatabase.csv')  
  
d =  
  
    userdata: {'My database'    '23-Sep-2011 14:10:17'}  
           x: [4x1 tseries]  
           y: [4x1 tseries]
```

the database will preserve the 'userdata=' field.

*Example 2* To change the field name under which you store your own user data, use the 'userdata=' option when running dbsave,

```
d = struct();
d.x = tseries(qq(2010,1):qq(2010,4),@rand);
d.y = tseries(qq(2010,1):qq(2010,4),@rand);
d.MYUSERDATA = {'My database',datestr(now())};
dbsave(d,'mydatabase.csv',Inf,'userData=', 'MYUSERDATA');
```

The name of the user data field is also kept in the CSV file so that dbload works fine in this case, too, and returns a database identical to the saved one,

```
d = dbload('mydatabase.csv')

d =

    MYUSERDATA: {'My database'   '23-Sep-2011 14:10:17'}
              x: [4x1 tseries]
              y: [4x1 tseries]
```

---

## ■ dbsearchuserdata

Search database to find tseries by matching the content of their userdata fields

### Syntax

```
LIST = dbsearchuserdata(D, FIELD1, REGEXP1, FIELD2, REGEXP2, ...)
LIST = dbsearchuserdata(D, FLAG, FIELD1, REGEXP1, FIELD2, REGEXP2, ...)
```

### Input arguments

- D [ struct ] - Input database whose tseries fields will be searched.
- FLAG [ '-all' | '-any' ] - Specifies if all conditions or any one condition must be met for the series to pass the test; if not specified, '-all' is assumed.

- FIELD1, FIELD2, ... [ char ] - Names of fields in the userdata struct.
- REGEXP1, REGEXP2, ... [ char ] - Regular expressions against which the respective userdata fields will be matched.

### Output arguments

- LIST [ cellstr ] - Names of tseries that pass the test.
- D [ struct ] - Output database with only those tseries that pass the test.

### Description

For a successful match, the userdata must be a struct, and the tested fields must be text strings.

Use an equal sign, =, after the name of the userdata fields in FIELD1, FIELD2, etc. to request a case-insensitive match, and an equal-shart sign, =#, to indicate a case-sensitive match.

### Example

```
[list,dd] = dbsearchuserdata(d,'.DESC=', 'Exchange rate', '.SOURCE=#', 'IMF');
```

Each individual tseries object in the database D will be tested for two conditions:

- whether its user data is a struct including a field named DESC, and the field contains a string 'Exchange rate' in it (case insensitive, e.g. 'eXcHaNgE rAtE' will also be matched);
- whether its user data is a struct including a field named SOURCE, and the field contains a string 'IMF' in it (case sensitive, e.g. 'Imf' will not be matched).

All tseries object that pass both of these conditions are returned in the LIST and the output database D.

---

## ■ dbuserdatalov

List of values found in a specified user data field in tseries objects

## Syntax

LOV = dbuserdatalov(D, FIELD)

## Input arguments

- D [ struct ] - Input database whose tseries objects will be searched.
- FIELD [ char ] - Name of a userdata field whose values will be collected across all tseries objects.

## Output arguments

- LOV [ cellstr ] - List of values found in the field FIELD of all tseries objects; only char values (text strings) are included; each value is included only once in LOV.

## Description

## Example

Part V —  
Reporting and publishing

## 19 Report objects and functions

### New report

- `new` P353 - New report object.
- `copy` P341 - Create a copy of a report object.

### Compiling PDF report

- `publish` P354 - Compile PDF from report object.

### Top-level objects

- `table` P360 - Start new table in report.
- `figure` P343 - Start new figure in report.
- `matrix` P348 - Insert matrix, or numeric array, in report.
- `modelfile` P351 - Write formatted model file in report.
- `array` P337 - Insert an array with user data in report.
- `tex` P363 - Include L<sup>A</sup>T<sub>E</sub>X code or verbatim text input in report.

### Figure objects

- `graph` P345 - Add graph to figure.

### Table and graph objects

- `band` P339 - Add new data with lower and upper bounds to graph or table.
- `fanchart` P342 - Add fancharts to graphs.
- `series` P356 - Add new data to graph or table.
- `subheading` P359 - Enter subheading in table.
- `vline` P364 - Add vertical line to graph.
- `highlight` P346 - Highlight range in graph.

## Structuring reports

- [align](#) P336 - Vertically align the following K objects.
- [empty](#) P342 - Empty report object.
- [include](#) P347 - Include text or LaTeX input file in the report.
- [merge](#) P351 - Merge the content of two or more report objects.
- [pagebreak](#) P353 - Force page break.
- [section](#) P355 - Start new section in report.

## Getting on-line help on report functions

```
help report  
help report/function_name
```

## Generic options

The following generic options can be used on any of the report objects.

- 'saveAs=' [ char | [empty](#) ] - (Not inheritable from parent objects) Save the LaTeX code generated for the respective report element in a text file under the specified name.

---

## ■ align

Vertically align the following K objects

### Syntax

```
P.align('',K,NCOL)
```

### Input arguments

- P [ struct ] - Report object created by the [report.new](#) P353 function.



- `K` [ numeric ] - Number of objects following this align that will be vertically aligned.
- `NCOL` [ numeric ] - Number of columns in which the objects will vertically aligned.

### Options

- `'hspace='` [ numeric | 2 ] - Horizontal space (in em units) inserted between two neighbouring objects.
- `'separator='` [ char | `'\medskip\par'` ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X commands that will be inserted after the aligned objects.
- `'typeface='` [ char | empty ] - (Not inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X code specifying the typeface for the align element as a whole; it must use the declarative forms (such as `\itshape`) and not the command forms (such as `\textit{...}`).

### Description

Vertically aligned can be the following types of objects:

- [figure](#) P343
- [table](#) P360
- [matrix](#) P348
- [array](#) P337

Note that the align object itself has no caption (even if you specify one it will not be used). Only the objects within align will be given captions. If the objects aligned on one row have identical captions (i.e. both titles and subtitles), only one caption will be displayed centred above the objects.

Because [empty](#) P342 objects count in the total number of objects included in align, you can use [empty](#) P342 in to create blank space in a particular position.

### Example

---

#### ■ array

Insert an array with user data in report

## Syntax

`P.array(CAP,DATA)`

## Input arguments

- `P` [ struct ] - Report object created by the [report.new](#) P353 function.
- `CAP` [ char | cellstr ] - Caption (title, subtitle) displayed at the top of the matrix.
- `DATA` [ cell ] - Cell array with input data; numeric and text entries are allowed.

## Options

- `'arrayStretch='` [ numeric | 1.15 ] - (Inheritable from parent objects) Stretch between lines in the array (in pts).
- `'captionTypeface='` [ cellstr | char | '\large\bfseries' ] - L<sup>A</sup>T<sub>E</sub>X format commands for typesetting the array caption; the subcaption format can be entered as the second cell in a cell array.
- `'colWidth'` [ numeric | NaN ] - (Inheritable from parent objects) Width of the array columns; if NaN the column width will be automatically adjusted.
- `'format='` [ char | '%.2f' ] - (Inheritable from parent objects) Numeric format string; see help on the built-in `sprintf` function.
- `'footnote='` [ char | empty ] - Footnote at the array title; only shows if the title is non-empty.
- `'heading='` [ char | empty ] - (Inheritable from parent objects) User-supplied heading, i.e. an extra row or rows at the top of the array.
- `'inf='` [ char | '\$\infty\$' ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X string that will be used to typeset Infs.
- `'long='` [ true | false ] - (Inheritable from parent objects) If true, the array may stretch over more than one page.
- `'longFoot='` [ char | empty ] - (Inheritable from parent objects) Footnote that appears at the bottom of the array (if it is longer than one page) on each page except the last one; works only with `'long=' true`.
- `'longFootPosition='` [ 'centre' | 'left' | 'right' ] - (Inheritable from parent objects) Horizontal alignment of the footnote in long arrays; works only with `'long=' true`.

- 'nan=' [ char | '\$\cdots\$' ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X string that will be used to typeset NaNs.
- 'pureZero=' [ char | empty ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- 'printedZero=' [ char | empty ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.
- 'separator=' [ char | '\medskip\par' ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X commands that will be inserted after the array.
- 'sideways=' [ true | false ] - (Inheritable from parent objects) Print the array rotated by 90 degrees.
- 'tabcolsep=' [ NaN | numeric ] - (Inheritable from parent objects) Space between columns in the array, measured in em units; NaN means the L<sup>A</sup>T<sub>E</sub>X default.
- 'typeface=' [ char | empty ] - (Not inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X code specifying the typeface for the array as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

---

## ■ band

Add new data with lower and upper bounds to graph or table

## Syntax

```
P.series(CAP,X,LOW,HIGH,...)
```

### Input arguments

- `P [ struct ]` - Report object created by the [report.new](#) P353 function.
- `CAP [ char ]` - Caption used as a default legend entry in a graph, or in the leading column in a table.
- `X [ tseries ]` - Input data with the centre of the band.
- `LOW [ tseries ]` - Input data with lower bounds; can be specified either relative to the centre or absolute, see the option `'relative='`.
- `HIGH [ tseries ]` - Input data with upper bounds; can be specified either relative to the centre or absolute, see the option `'relative='`.

### Options for table and graph bands

- `'low=' [ char | 'Low' ]` - (Inheritable from parent objects) Mark used to denote the lower bound.
- `'high=' [ char | 'High' ]` - (Inheritable from parent objects) Mark used to denote the upper bound.
- `'relative=' [ true | false ]` - (Inheritable from parent objects) If true, the data for the lower and upper bounds are relative to the centre, i.e. the bounds will be added to the centre (in this case, LOW must be negative numbers and HIGH must be positive numbers). If false, the bounds are absolute data (in this case LOW must be lower than X, and HIGH must be higher than X).

### Options for table bands

- `'bandTypeface=' [ char | '\footnotesize' ]` - (Inheritable from parent objects) LaTeX format string used to typeset the lower and upper bounds.%

### Options for graph bands

- `'plotType=' [ 'errorbar' | 'patch' ]` - Type of plot used to draw the band.
- `'relative=' [ true | false ]` - (Inheritable from parent objects) If true the lower and upper bounds will be, respectively, subtracted from and added to to the middle line.
- `'white=' [ numeric | 0.8 ]` - (Inheritable from parent objects) Proportion of white colour mixed with the line colour and used to fill the area that depicts the band.

See help on [report/series](#) P356 for other options available.

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

---

### ■ copy

Create a copy of a report object

## Syntax

```
Q = copy(P)
```

## Input arguments

- P [ report ] - Report object whose copy will be created.

## Output arguments

- Q [ report ] - Copy of the input report object.

## Description

Because report is a handle class object, a plain assignment

```
Q = P;
```

creates a handle to the same copy of a report object. In other words, changes in Q will also change P and vice versa. To make a new, independent copy of an existing report object, you need to run

```
Q = copy(P);
```

---

## ■ empty

Empty report object

### Syntax

```
P.empty()
```

### Input arguments

- P [ struct ] - Report object created by the [report.new](#) P353 function.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

The empty object does not produce any visible output in the report. It can be used in [align](#) P336 or [figure](#) P343 to create blank space.

### Example

---

## ■ fanchart

Add fancharts to graphs

### Syntax

```
P.fanchart(CAP,X,STD,PROB,...)
```

### Input arguments

- P [ struct ] - Report object created by the [report.new](#) P353 function.

- CAP [ char ] - Caption used as a legend entry for the line (mean of fanchart)
- X [ tseries ] - Tseries object with input data to be displayed.
- STD [ tseries ] - Tseries object with standard deviations of input data.
- PROB [ numeric ] - Confidence probabilities of intervals to be displayed.

### Options for fancharts

- 'asym=' [ numeric | 1 ] - Ratio of asymmetry (area of upper part to one of lower part).
- 'exclude=' [ numeric | true | false ] - Exclude some of the confidence intervals.
- 'factor=' [ numeric | 1 ] - factor to increase or decrease input standard deviations
- 'fanlegend=' [ cell | NaN | Inf ] - Legend entries used instead of confidence interval values; Inf means all confidence intervals values will be used to construct legend entries; NaN means the intervals will be excluded from legend; NaN in cellstr means the intervals of respective fancharts will be excluded from legend.

See help on [report/series](#) P356 for other options available.

### Description

The confidence intervals are based on normal distributions with standard deviations supplied by the user. Optionally, the user can also specify assumptions about asymmetry and/or common correction factors.

### Example

---

## ■ figure

Start new figure in report

### Syntax

```
P.figure(CAPTION,...)
P.figure(CAPTION,FIG)
```

## Input arguments

- `P` [ struct ] - Report object created by the [report.new](#) P353 function.
- `CAPTION` [ char | cellstr ] - Title and subtitle displayed at the top of the figure.
- `FIG` [ numeric ] - Handle to a graphics figure created by the user that will be captured and inserted in the report; the report figure object must have no children-graphs in this case; if no handle is specified, the current figure will be captured.

## Options

- `'captionTypeface='` [ cellstr | char | `'\large\bfseries'` ] - LaTeX format commands for typesetting the figure caption; the subcaption format can be entered as the second cell in a cell array.
- `'close='` [ `true` | `false` ] - (Inheritable from parent objects) Close Matlab the underlying figure window when finished.
- `'separator='` [ char | `'\medskip\par'` ] - (Inheritable from parent objects) LaTeX commands that will be inserted after the figure.
- `'figureOptions='` [ cell | `empty` ] - Figure options that will be applied to the figure handle after all graphs are drawn.
- `'figureScale='` [ numeric | `0.85` ] - (Inheritable from parent objects) Scale of the figure in the LaTeX document.
- `'footnote='` [ char | `empty` ] - Footnote at the figure title; only shows if the title is non-empty.
- `'sideways='` [ `true` | `false` ] - (Inheritable from parent objects) Print the table rotated by 90 degrees.
- `'style='` [ struct | `empty` ] - Apply this cascading style structure to the figure; see [qstyle](#) P374.
- `'subplot='` [ numeric | `'auto'` ] - (Inheritable from parent objects) Subplot division of the figure.
- `'typeface='` [ char | `empty` ] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the figure as a whole; it must use the declarative forms (such as `\itshape`) and not the command forms (such as `\textit{...}`).
- `'visible='` [ `true` | `false` ] - (Inheritable from parent objects) Visibility of the underlying Matlab figure window.



## Generic options

See help on [generic options](#) [P335](#) in report objects.

## Description

Figures are top-level report objects and cannot be nested within other report objects, except [align](#) [P336](#). Figure objects can have the following types of children:

- [graph](#) [P345](#);
- [empty](#) [P342](#).

## Example

---

## ■ graph

Add graph to figure

## Syntax

P.graph(CAPTION,...)

## Input arguments

- P [ struct ] - Report object created by the [report.new](#) [P353](#) function.
- CAPTION [ char | cellstr ] - Title, or cell array with title and subtitle, displayed at the top of the graph.

## Options

- 'axesOptions=' [ cell | [empty](#) ] - Options executed by calling set on the axes handle before running 'postProcess='.
- 'dateFormat=' [ char | 'YYYY:P' ] - (Inheritable from parent objects) Date format string, see help on [dat2str](#) [P231](#).

## Report objects and functions: highlight

- 'dateTick=' [ numeric | Inf ] - (Inheritable from parent objects) Date tick spacing.
- 'legend=' [ false | true ] - (Inheritable from parent objects) Add legend to the graph.
- 'legendLocation=' [ char | 'best' ] - (Inheritable from parent objects) Location of the legend box; see help on legend for values available.
- 'postProcess=' [ char | empty ] - String with Matlab commands executed after the graph has been drawn and styled; the commands have access to variable H, a handle to the current axes object.
- 'preProcess=' [ char | empty ] - String with Matlab commands executed before the graph has been drawn and styled; the commands have access to variable H, a handle to the current axes object.
- 'range=' [ numeric | Inf ] - (Inheritable from parent objects) Graph range.
- 'style=' [ struct | empty ] - (Inheritable from parent objects) Apply this style structure to the graph and its children; see help on [qstyle](#) P374.
- 'tight=' [ true | false ] - (Inheritable from parent objects) Set the y-axis limits to the minimum and maximum of displayed data.
- 'zeroLine=' [ true | false ] - (Inheritable from parent objects) Add a horizontal zero line if zero is included on the y-axis.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

### Example

---

## ■ highlight

Highlight range in graph

### Syntax

P.highlight(CAP,RANGE,...)

Report objects and functions: include

### Input arguments

- `P [ struct ]` - Report object created by the [report.new](#) P353 function.
- `CAP [ char ]` - Caption used to annotate the highlighted area.
- `RANGE [ cell | numeric ]` - Date range, or a cell array of ranges, that will be highlighted.

### Options

- `'hPosition=' [ 'bottom' | 'middle' | 'top' ]` - (Inheritable from parent objects) Horizontal position of the caption.
- `'vPosition=' [ 'centre' | 'left' | 'right' ]` - (Inheritable from parent objects) Vertical position of the caption relative to the edges of the highlighted area.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

### Example

---

## ■ include

Include text or LaTeX input file in the report

### Syntax

```
P.include(CAP,FNAME,...)
```

### Input arguments

- `P [ struct ]` - Report object created by the [report.new](#) P353 function.
- `CAP [ char ]` - Caption displayed at the top of the file included.
- `FNAME [ char ]` - File name that will be included here.

## Options

- `'centering='` [ `true` | `false` ] - (Inheritable from parent objects) Centre the content of the file on the page.
- `'separator='` [ `char` | `empty` ] - (Not inheritable from parent objects)  $\text{\LaTeX}$  commands that will be inserted after the table.
- `'typeface='` [ `char` | `empty` ] - (Not inheritable from parent objects)  $\text{\LaTeX}$  code specifying the typeface for the include element as a whole; it must use the declarative forms (such as `\itshape`) and not the command forms (such as `\textit{...}`).
- `'verbatim='` [ `true` | `false` ] - (Not inheritable from parent objects) Enclose the content of the file in a verbatim environment.

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

---

## ■ `matrix`

Insert matrix, or numeric array, in report

## Syntax

```
P.matrix(CAP,DATA,...)
```

## Input arguments

- `P` [ `struct` ] - Report object created by the [report.new](#) P353 function.
- `CAP` [ `char` | `cellstr` ] - Caption (title, subtitle) displayed at the top of the matrix.
- `DATA` [ `numeric` ] - Numeric array with input data.

## Options

- 'arrayStretch=' [ numeric | 1.15 ] - (Inheritable from parent objects) Stretch between lines in the matrix (in pts).
- 'captionTypeface=' [ cellstr | char | ' ] -  $\LaTeX$  format commands for typesetting the matrix caption; the subcaption format can be entered as the second cell in a cell array.
- 'colNames=' [ cellstr | empty ] - (Inheritable from parent objects) Names for individual matrix columns, displayed at the top of the matrix.
- 'colWidth=' [ numeric | NaN ] - (Inheritable from parent objects) Width of the matrix columns; NaN means the column width will be automatically adjusted.
- 'condFormat=' [ struct | empty ] - (Inheritable from parent objects) Structure with .test and .format fields describing conditional formatting of individual matrix entries.
- 'footnote=' [ char | empty ] - Footnote at the matrix title; only shows if the title is non-empty.
- 'format=' [ char | '%.2f' ] - (Inheritable from parent objects) Numeric format string; see help on the built-in sprintf function.
- 'heading=' [ char | empty ] - (Inheritable from parent objects) User-supplied heading, i.e. an extra row or rows at the top of the matrix.
- 'inf=' [ char | '\$\infty\$' ] - (Inheritable from parent objects)  $\LaTeX$  string that will be used to typeset Infs.
- 'long=' [ true | false ] - (Inheritable from parent objects) If true, the matrix may stretch over more than one page.
- 'longFoot=' [ char | empty ] - (Inheritable from parent objects) Works only with 'long=' true: Footnote that appears at the bottom of the matrix (if it is longer than one page) on each page except the last one.
- 'longFootPosition=' [ 'centre' | 'left' | 'right' ] - (Inheritable from parent objects) Works only with 'long=' true: Horizontal alignment of the footnote in long matrices.
- 'nan=' [ char | '\$\cdots\$' ] - (Inheritable from parent objects)  $\LaTeX$  string that will be used to typeset NaNs.
- 'pureZero=' [ char | empty ] - (Inheritable from parent objects)  $\LaTeX$  string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- 'printedZero=' [ char | empty ] - (Inheritable from parent objects)  $\LaTeX$  string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.

- 'rowNames=' [ cellstr | empty ] - (Inheritable from parent objects) Names for individual matrix rows, displayed left of the matrix.
- 'separator=' [ char | '\medskip\par' ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X commands that will be inserted after the matrix.
- 'sideways=' [ true | false ] - (Inheritable from parent objects) Print the matrix rotated by 90 degrees.
- 'tabcolsep=' [ NaN | numeric ] - (Inheritable from parent objects) Space between columns in the matrix, measured in em units; NaN means the L<sup>A</sup>T<sub>E</sub>X default.
- 'typeface=' [ char | empty ] - (Not inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X code specifying the typeface for the matrix as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

*Conditional formatting* The conditional format struct (or an array of structs) specified through the 'condFormat=' option must have two fields, .test and .format.

The .test field is a text string with a Matlab expression. The expression must evaluate to a scalar true or false, and can refer to the following attributes associated with each entry in the data part of the matrix:

- value - the numerical value of the entry;
- row - the row number within the data part of the matrix;
- col - the col number within the data part of the matrix;
- rowname - the row name right of which the entry appears;
- colname - the column name under which the entry appears;
- rowvalues - a row vector of all values in the current row;
- colvalues - a column vector of all values in the current column;
- allvalues - a matrix of all values.

You can combine a number of attributes within one test, using the logical operators, e.g.

```
value > 0 && row > 3
value == max(rowvalues) && strcmp(rowname,'x')
```

The `.format` fields of the conditional format structure consist of  $\text{\LaTeX}$  commands that will be used to typeset the corresponding entry. The reference to the entry itself is through a question mark. The entries are typeset in math mode; this for instance means that for bold or italic typface, you must use the `\mathbf{\{...\}}` and `\mathit{\{...\}}` commands.

Furthermore, you can combine multiple tests and their corresponding formats in one structure; they will be all applied to each entry in the specified order.

### Example

---

## ■ merge

Merge the content of two or more report objects

### Syntax

```
P.merge(P1,P2,...)
```

### Input arguments

- `P [ report ]` - Report object created by the [report.new](#) P353 function.
- `P1, P2 [ report ]` - Other report objects whose content will be added to `P`.

### Description

### Example

---

## ■ modelfile

Write formatted model file in report

## Syntax

```
P.modelfile(CAP,FILENAME,...)
P.modelfile(CAP,FILENAME,M,...)
```

## Input arguments

- `P [ report ]` - Report object created by the `report.new` P353 function.
- `CAP [ char | cellstr ]` - Title and subtitle displayed at the top of the table.
- `FILENAME [ char ]` - Model file name.
- `M [ model ]` - Model object from which the values of parameters and std devs of shocks will be read; if missing no parameter values or std devs will be printed.

## Options

- `'lines=' [ numeric | Inf ]` - Print only selected lines of the model file `FILENAME`; `Inf` means all lines will be printed.
- `'lineNumbers=' [ true | false ]` - Display line numbers.
- `'footnote=' [ char | empty ]` - Footnote at the model file title; only shows if the title is non-empty.
- `'paramValues=' [ true | false ]` - Display the values of parameters and std devs of shocks next to each occurrence of a parameter or a shock; this option works only if a model object `M` is entered as the 3rd input argument.
- `'syntax=' [ true | false ]` - Highlight model file syntax; this includes model language keywords, descriptions of variables, shocks and parameters, and equation labels.
- `'typeface=' [ char | empty ]` - (Not inheritable from parent objects) LaTeX code specifying the typeface for the model file as a whole; it must use the declarative forms (such as `\itshape`) and not the command forms (such as `\textit{...}`).

## Description

If you enter a model object with multiple parameterisations, only the first parameterisation will get reported.

At the moment, the syntax highlighting in model file reports does not handle correctly comment blocks, i.e. `%{ ... %}`.



## Example

---

### ■ new

New report object

#### Syntax

```
x = report.new(CAP,...)
```

#### Output arguments

- x [ struct ] - Report object with function handles through which the individual report elements can be created.
- CAP [ char ] - Report caption; the caption will also be printed on the title page of the report if published with the option 'makeTitle=' true.

#### Options

- 'centering=' [ true | false ] - All report elements, except [tex](#) P363, will be centered on the page.
- 'orientation=' [ 'landscape' | 'portrait' ] - Paper orientation of the published report.

Report options are cascading. You can specify any of an object's options in any of his parent (or ascendant) objects.

---

### ■ pagebreak

Force page break

#### Syntax

```
P.pagebreak(...)
```

### Input arguments

- `P [ report ]` - Report object created by the `report.new` P353 function.

### Generic options

See help on `generic options` P335 in report objects.

### Description

### Example

---

## ■ publish

Compile PDF from report object

### Syntax

```
P.publish(FNAME,...)
```

### Input arguments

- `P [ struct ]` - Report object created by the `report.new` function.
- `FNAME [ char ]` - File name under which the compiled PDF will be saved.

### Options

- `'abstract=' [ char | empty ]` - Abstract on the title page.
- `'author=' [ char | empty ]` - List of authors on the title page separated with `\and` or `\\`.
- `'date=' [ char | '\today' ]` - Date on the title page.
- `'cleanup=' [ true | false ]` - Delete all temporary files created when compiling the report.
- `'epsToPdf=' [ char | Inf ]` - Command line arguments for EPSTOPDF; if `Inf` system-dependent defaults are used.

- 'fontEnc=' [ char | 'T1' ] - L<sup>A</sup>T<sub>E</sub>X font encoding.
- 'makeTitle=' [ true | false ] - Produce title page (with title, author, date, and abstract).
- 'paperSize=' [ 'a4paper' | 'letterpaper' ] - Paper size.
- 'orientation=' [ 'landscape' | 'portrait' ] - Paper orientation.
- 'preamble=' [ char | empty ] - L<sup>A</sup>T<sub>E</sub>X commands that will be placed in the L<sup>A</sup>T<sub>E</sub>X file preamble.
- 'title=' [ char | Inf ] - Report title on the first page; Inf means the report caption specified in the [report.new](#) P353 function will be used.
- 'timeStamp=' [ char | 'datestr(now())' ] - String printed in the top-left corner of each page.
- 'maxRerun=' [ numeric | 5 ] - Maximum number of times the L<sup>A</sup>T<sub>E</sub>X compiler will be run to resolve cross-references, etc.
- 'scale=' [ numeric | 0.8 ] - Percentage of the total page area that will be used.

## Description

## Example

---

## ■ section

Start new section in report

## Syntax

```
p.section(CAP,...)
```

## Input arguments

- P [ struct ] - Report object created by the [report.new](#) P353 function.
- CAP [ char ] - Section title.

## Options

- 'numbered=' [ `true` | `false` ] - (Inheritable from parent objects) Numbered section.
- 'separator=' [ `char` | `empty` ] - (Not inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X commands that will be inserted after the table.

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

# ■ series

Add new data to graph or table

## Syntax

```
P.series(CAP,X,...)
```

## Input arguments

- P [ `struct` ] - Report object created by the [report.new](#) P353 function.
- CAP [ `char` ] - Caption used as a default legend entry in a graph, or in the leading column in a table.
- X [ `tseries` ] - Input data that will be added to the current table or graph.

## Options for both table series and graph series

- 'marks=' [ `cellstr` | `empty` ] - (Inheritable from parent objects) Marks that will be added to the legend entries in graphs, or printed in a third column in tables, to distinguish the individual columns of possibly multivariate input `tseries` objects.

- 'showMarks=' [ true | false ] - (Inheritable from parent objects) Use the marks defined in the 'marks=' option to label the individual rows when input data is a multivariate tseries object.

### Options for table series

- 'autoData=' [ function\_handle | cell | empty ] - Function, or a cell array of functions, that will be used to produce new columns in the input tseries object (i.e. new rows of output in the report).
- 'condFormat=' [ struct | empty ] - (Inheritable from parent objects) Structure with .test and .format fields describing conditional formatting of individual table entries.
- 'decimal=' [ numeric | NaN ] - (Inheritable from parent objects) Number of decimals that will be displayed; if NaN the 'format=' option is used instead.
- 'format=' [ char | '%.2f' ] - (Inheritable from parent objects) Numeric format string; see help on the built-in sprintf function.
- 'footnote=' [ char | empty ] - Footnote at the series text.
- 'highlight=' [ numeric | empty ] - (Inheritable from parent objects) Periods that will get highlighted in tables; to highlight date ranges in graphs, use the 'highlight=' option in the parent graph object.
- 'inf=' [ char | '\ensuremath{\infty}' ] - (Inheritable from parent objects) LaTeX string that will be used to typeset Inf entries.
- 'nan=' [ char | '\ensuremath{\cdot}' ] - (Inheritable from parent objects) LaTeX string that will be used to typeset NaN entries.
- 'pureZero=' [ char | empty ] - (Inheritable from parent objects) LaTeX string that will be used to typeset pure zero entries; if empty the zeros will be printed using the current numeric format.
- 'printedZero=' [ char | empty ] - (Inheritable from parent objects) LaTeX string that will be used to typeset the entries that would appear as zero under the current numeric format used; if empty these numbers will be printed using the current numeric format.
- 'separator=' [ char | empty ] - (Not inheritable from parent objects) LaTeX commands that will be inserted immediately after the end of the table row, i.e. appended to \, within a tabular mode.
- 'units=' [ char ] - (Inheritable from parent objects) Description of input data units that will be displayed in the second column of tables.

### Options for graph series

- 'legend=' [ char | cellstr | NaN | Inf ] - (Not inheritable from parent objects) Legend entries used instead of the series caption and marks; Inf means the caption and marks will be used to construct legend entries; NaN means the series will be excluded from legend.
- 'plotFunc=' [ @area | @bar | @barcon | @plot | @plotcmp | @plotpred | @stem ] - (Inheritable from parent objects) Plot function that will be used to create graphs.
- 'plotOptions=' [ cell | empty ] - Options passed as the last input arguments to the plot function.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

*Using the 'nan=', 'inf=', 'pureZero=' and 'printedZero=' options* When specifying the LaTeX string for these options, bear in mind that the table entries are printed in the math model. This means that whenever you wish to print a normal text, you need to use an appropriate text formatting command allowed within a math mode. Most frequently, it would be '`\textnormal{...}`'.

*Using the 'plotFunc=' option* When you set the option to 'plotpred', the input data X (second input argument) must be a multicolumn tseries object where the first column is the time series observations, and the second and further columns are its Kalman filter predictions as returned by the filter function.

*Conditional formatting* The conditional format struct (or an array of structs) specified through the 'condFormat=' option must have two fields, .test and .format.

The .test field is a text string with a Matlab expression. The expression must evaluate to a scalar true or false, and can refer to the following attributes associated with each entry in the data part of the table:

- value - the numerical value of the entry,
- date - the date under which the entry appears,
- year - the year under which the entry appears,
- period - the period within the year (e.g. month or quarter) under which the entry appears,

- freq - the frequency of the date under which the entry appears,
- text - the text label on the left,
- mark - the text mark on the left used to describe the individual rows reported for multivariate series,
- row - the row number within a multivariate series.
- rowvalues - a row vector of all values on the current row.

If the table is based on user-defined structure of columns (option 'colstruct=' in `table` P360), the following additional attributes are available

- colname - descriptor of the column (text in the headline).

You can combine a number of attributes within one test, using the logical operators, e.g.

```
'value > 0 && year > 2010'
```

The `.format` fields of the conditional format structure consist of LaTeX commands that will be used to typeset the corresponding entry. The reference to the entry itself is through a question mark. The entries are typeset in math mode; this for instance means that for bold or italic typface, you must use the `\mathbf{...}` and `\mathit{...}` commands.

Furthermore, you can combine multiple tests and their corresponding formats in one structure; they will be all applied to each entry in the specified order.

### Example of a conditional format structure

```
cf = struct();
cf(1).test = 'value < 0';
cf(1).format = '\mathit{?}';
cf(2).test = 'date < qq(2010,1)';
cf(2).format = '\color{blue}';
```

---

## ■ subheading

Enter subheading in table

## Syntax

P.subheading(CAP,...)

## Input arguments

- P [ struct ] - Report object created by the [report.new](#) P353 function.
- CAP [ char ] - Text displayed as a subheading on a separate line in the table.

## Options

- 'justify=' [ 'c' | 'l' | 'r' ] - (Inheritable from parent objects) Horizontal alignment of the subheading (centre, left, right).
- 'separator=' [ char | empty ] - (Not inheritable from parent objects) LaTeX commands that will be inserted immediately after the end of the table row, i.e. appended to \, within a tabular mode.
- 'stretch=' [ true | false ] - (Inheritable from parent objects) Stretch the subheading text also across the data part of the table; if not the text will be contained within the initial descriptive columns.
- 'typeface=' [ char | '\itshape\bfseries' ] - (Not inheritable from parent objects) LaTeX code specifying the typeface for the subheading; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

---

## ■ table

Start new table in report



## Syntax

`P.table(CAP,...)`

## Input arguments

- `P [ report ]` - Report object created by the `report.new` P353 function.
- `CAP [ char | cellstr ]` - Title and subtitle displayed at the top of the table.

## Options

- `'arrayStretch=' [ numeric | 1.15 ]` - (Inheritable from parent objects) Stretch between lines in the table (in pts).
- `'captionTypeface=' [ cell | '\large\bfseries' ]` - LaTeX format commands for typesetting the table caption and subcaption; you can use `Inf` for either to indicate the default format.
- `'colStruct=' [ struct | empty ]` - (Not inheritable from parent objects) User-defined structure of the table columns; use of this option disables `'range='`.
- `'colWidth=' [ numeric | NaN ]` - (Inheritable from parent objects) Width of table columns in em units; if `NaN` the width of each column will adjust automatically.
- `'headlineJustify=' [ 'c' | 'l' | 'r' ]` - Horizontal alignment of the headline entries (individual dates or user-defined text): Centre, Left, Right.
- `'dateFormat=' [ char | cellstr | irisget('dateformat') ]` - (Inheritable from parent objects) Format string for the date row.
- `'footnote=' [ char | empty ]` - Footnote at the table title; only shows if the title is non-empty.
- `'long=' [ true | false ]` - (Inheritable from parent objects) If true, the table may stretch over more than one page.
- `'longFoot=' [ char | empty ]` - (Inheritable from parent objects) Works only with `'long'=true`: Footnote that appears at the bottom of the table (if it is longer than one page) on each page except the last one.
- `'longFootPosition=' [ 'centre' | 'left' | 'right' ]` - (Inheritable from parent objects) Works only with `'long=' true`: Horizontal alignment of the footnote in long tables.
- `'range=' [ numeric | empty ]` - (Inheritable from parent objects) Date range or vector of dates that will appear as columns of the table.

- 'separator=' [ char | '\medskip\par' ] - (Inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X commands that will be inserted after the table.
- 'sideways=' [ true | false ] - (Inheritable from parent objects) Print the table rotated by 90 degrees.
- 'tabcolsep=' [ NaN | numeric ] - (Inheritable from parent objects) Space between columns in the table, measured in em units; NaN means the L<sup>A</sup>T<sub>E</sub>X default.
- 'typeface=' [ char | empty ] - (Not inheritable from parent objects) L<sup>A</sup>T<sub>E</sub>X code specifying the typeface for the table as a whole; it must use the declarative forms (such as \itshape) and not the command forms (such as \textit{...}).
- 'vline=' [ numeric | empty ] - (Inheritable from parent objects) Vector of dates after which a vertical line (divider) will be placed.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

Tables are top-level report objects and cannot be nested within other report objects, except [align](#) P336. Table objects can have the following children:

- [series](#) P356;
- [subheading](#) P359.

By default, the date row is printed as a leading row with dates formatted using the option 'dateFormat='. Alternatively, you can specify this option as a cell array of two strings. In that case, the dates will be printed in two rows. The first row will have a date string displayed and centred for every year, and the first cell of the 'dateFormat=' option will be used for formatting. The second row will have a date displayed for every period (i.e. every column), and the second cell of the 'dateFormat=' option will be used for formatting.

*User-defined structure of the table columns* Use can use the 'columnStruct=' option to define your own table columns. This gives you more flexibility than when using the 'range=' option in defining the content of the table.

The option 'columnStruct=' must be a 1-by-N struct, where N is the number of columns you want in the table, with the following fields:

- 'name=' - specifies the descriptor of the column that will be displayed in the headline;
- 'func=' - specifies a function that will be applied to the input series; if 'func=' is empty, no function will be applied. The function must evaluate to a tseries or a numeric scalar.
- 'date=' - specifies the date at which a number will be taken from the series unless the function 'func=' applied before resulted in a numeric scalar.

### Example

Compare the headers of these two tables:

```
x = report.new();

x.table('First table', ...
        'range',qq(2010,1):qq(2012,4), ...
        'dateformat','YYYYFP');
% You can add series or subheadings here.

x.table('Second table', ...
        'range',qq(2010,1):qq(2012,4), ...
        'dateformat',{'YYYY','FP'});
% You can add series or subheadings here.

x.publish('myreport.pdf');
```

---

## ■ tex

Include  $\text{\LaTeX}$  code or verbatim text input in report

Syntax with input specified in comment block

```
P.tex(CAP,...)

%{
Write text or \LaTeX\ code as a block comment
right after the p.text(...) command.
%}
```

### Syntax with input specified as `char` argument

```
P.tex(CAP, CODE, ...)
```

### Input arguments

- `P` [ `struct` ] - Report object created by the [report.new](#) P353 function.
- `CAP` [ `char` ] - Caption displayed at the top of the text.
- `CODE` [ `char` ] -  $\text{\LaTeX}$  code or text input that will be included in the report.

### Options

- `'centering='` [ `true` | `false` ] - (Inheritable from parent objects) Centre the  $\text{\LaTeX}$  code or text input on the page.
- `'footnote='` [ `char` | `empty` ] - Footnote at the tex block title; only shows if the title is non-empty.
- `'separator='` [ `char` | `'\medskip\par'` ] - (Inheritable from parent objects)  $\text{\LaTeX}$  commands that will be inserted after the text.
- `'verbatim='` [ `true` | `false` ] - If true the text will be typeset verbatim in monospaced font; if false the text will be treated as  $\text{\LaTeX}$  code included in the report.

### Generic options

See help on [generic options](#) P335 in report objects.

### Description

### Example

---

## ■ `vline`

Add vertical line to graph

## Syntax

`P.vline(CAP,DATE,...)`

## Input arguments

- `P` [ struct ] - Report object created by the [report.new](#) P353 function.
- `CAP` [ char ] - Caption used to annotate the vertical line.
- `DATE` [ numeric ] - Date at which the vertical line will be plotted.

## Options

- `'hPosition='` [ 'bottom' | 'middle' | 'top' ] - (Inheritable from parent objects) Horizontal position of the caption.
- `'vPosition='` [ 'centre' | 'left' | 'right' ] - (Inheritable from parent objects) Vertical position of the caption relative to the line.
- `'timePosition='` [ 'after' | 'before' | 'middle' ] - Placement of the vertical line on the time axis: in the middle of the specified period, immediately before it (between the specified period and the previous one), or immediately after it (between the specified period and the next one).

## Generic options

See help on [generic options](#) P335 in report objects.

## Description

## Example

## 20 Quick-report file language

### Figures

- `!++` P368 - Create new figure window.
- `#MxN` P370 - Specify the subplot division of the figure windows. figure windows.

### Graphs

- `!--` P369 - Create new line graph.
- `!::` P366 - Create new bar graph.
- `!ii` P369 - Create new stem graph.
- `!II` P367 - Create new errorbar graph.
- `!^^` P368 - Create new histogram.
- `!..` P367 - Skip the current subplot position leaving it blank.

### Formatting graph titles

- `//` P369 - Line break in graph title.
- `--` P370 - Subtitle in graph title.

### Getting on-line help on qreport file language

```
help qreportlang  
help qreportlang/keyword
```

---

### ■ !::

Create new bar graph

### Syntax

```
!:: graph_caption  
    expression1, expression2, ...
```

### Description

### Example

---

#### ■ !..

Skip the current subplot position leaving it blank

### Syntax

```
!..
```

### Description

### Example

---

#### ■ !II

Create new errorbar graph

### Syntax

```
!II TITLE  
    EXPRESSION1, EXPRESSION2, ...
```

### Description

### Example

---

■ !++

Create new figure window

### Syntax

```
!++ figure_caption
```

### Description

### Example

-IRIS Toolbox. -Copyright (c) 2007–2012 Jaromir Benes.

---

■ !^^

Create new histogram

### Syntax

```
!^^ graph_caption  
    expression1, expression2, ...
```

### Description

### Example

---



## ■ //

Line break in graph title

### Syntax

```
!-- FIRST LINE // SECOND LINE // ...
```

### Description

### Example

---

## ■ !—

Create new line graph

### Syntax

```
!-- graph_caption  
    expression1, expression2, ...
```

### Description

### Example

---

## ■ !ii

Create new stem graph

### Syntax

```
!ii TITLE  
    EXPRESSION1, EXPRESSION2, ...
```

## Description

## Example

---

### ■ #MxN

Specify the subplot division of the figure windows

#### Syntax with user-supplied subdivision

#MxN

#### Syntax for automatic subdivision

#auto

#### Input arguments

- M [ numeric ] - Number of rows of graphs in each figure window from this point on.
- N [ numeric ] - Number of columns of graphs in each figure window from this point on.

## Description

## Example

This command produces figure windows with 3x2 graphs in each.

#3x2

---

### ■ \_ \_

Subtitle in graph title

Quick-report file language: \_\_

### **Syntax**

```
!-- TITLE __ SUBTITLE
```

### **Description**

### **Example**

## 21 Quick-report functions

### Quick-report functions

- [qplot](#) P372 - Quick report.
- [qstyle](#) P374 - Apply styles to graphics object and its descendants.

### Getting on-line help on qreport functions

```
help qreport  
help qreport/function_name
```

---

## ■ qplot

### Quick report

#### Syntax

```
[FF,AA,PDB] = qplot(QFILE,D,RANGE,...)
```

#### Input arguments

- QFILE [ char ] - Name of the q-file that defines the contents of the individual graphs.
- D [ struct ] - Database with input data.
- RANGE [ numeric ] - Date range.

#### Output arguments

- FF [ numeric ] - Handles to figures created by qplot.
- AA [ cell ] - Handles to axes created by qplot.
- PDB [ struct ] - Database with actually plotted series.

## Options

- 'addClick=' [ true | false ] - Make axes expand in a new graphics figure upon mouse click.
- 'clear=' [ numeric | empty ] - Serial numbers of graphs (axes objects) that will not be displayed.
- 'dbsave=' [ cellstr | empty ] - Options passed to dbsave when 'saveAs=' is used.
- 'drawNow=' [ true | false ] - Call Matlab drawnow function upon completion of all figures.
- 'grid=' [ true | false ] - Add grid lines to all graphs.
- 'highlight=' [ numeric | cell | empty ] - Date range or ranges that will be highlighted.
- 'interpreter=' [ 'latex' | 'none' ] - Interpreter used in graph titles.
- 'mark=' [ cellstr | empty ] - Marks that will be added to each legend entry to distinguish individual columns of multivariate tseries objects plotted.
- 'overflow=' [ true | false ] - Open automatically a new figure window if the number of subplots exceeds the available total; 'overflow' = false means an error will occur instead.
- 'prefix=' [ char | 'P%g\_' ] - Prefix (a sprintf format string) that will be used to precede the name of each entry in the PDB database.
- 'round=' [ numeric | Inf ] - Round the input data to this number of decimals before plotting.
- 'saveAs=' [ char | empty ] - File name under which the plotted data will be saved either in a CSV data file or a PDF; you can use the 'dbsave=' option to control the options used when saving CSV.
- 'style=' [ struct | empty ] - Style structure that will be applied to all figures and their children created by the qplot function.
- 'subplot=' [ 'auto' | numeric ] - Default subplot division of figures, can be modified in the q-file.
- 'sstate=' [ struct | model | empty ] - Database or model object from which the steady-state values referenced to in the quick-report file will be taken.
- 'style=' [ struct | empty ] - Style structure that will be applied to all created figures upon completion.
- 'transform=' [ function\_handle | empty ] - Function that will be used to transform
- 'title=' [ cellstr | @comment | empty ] - Strings that will be used for titles in the graphs that have no title in the q-file.
- 'tight=' [ true | false ] - Make the y-axis in each graph tight.

- 'vLine=' [ numeric | empty ] - Dates at which vertical lines will be plotted.
- 'zeroLine=' [ true | false ] - Add a horizontal zero line to graphs whose y-axis includes zero.

## Description

## Example

---

## ■ qstyle

Apply styles to graphics object and its descendants

## Syntax

```
qstyle(H,S,...)
```

## Input arguments

- H [ numeric ] - Handle to a graphics object that will be styled along with its descendants (unless 'cascade=' is false).
- S [ struct ] - Struct each field of which refers to an object-dot-property; the value of the field will be applied to the the respective property of the respective object; see below the list of graphics objects allowed.

## Options

- 'cascade=' [ true | false ] - Cascade through all descendants of the object H; if false only the object H itself will be styled.
- 'warning=' [ true | false ] - Display warnings produced by this function.

## Description

The style structure, S, is constructed of any number of nested object-property fields:

```
S.object.property = value;
```

The following is the list of standard Matlab graphics objects the first-level fields can refer to:

- figure
- axes
- title
- xlabel
- ylabel
- zlabel
- line
- bar
- patch
- text

In addition, you can also refer to the following special instances of objects created by IRIS functions:

- legend (an axes object)
- plotpred (line objects with prediction data created by plotpred)
- highlight (a patch object created by highlight)
- highlightcaption (a text object created by highlight)
- vline (a line object created by vline)
- vlinecaption (a text object created by vline)
- zeroline (a line object created by zeroline)

The property used as the second-level field is simply any regular Matlab property of the respective object (see Matlab help on graphics).

The value assigned to a particular property can be either of the following:

- a single proper valid value (i.e. a value you would be able to assign using the standard Matlab set function);
- a cell array of multiple different values that will be assigned to the objects of the same type in order of their creation;

### Quick-report functions: qstyle

- a text string starting with a double exclamation point, `!!`, followed by Matlab commands. The commands are expected to eventually create a variable named `SET` whose value will then assigned to the respective property. The commands have access to variable `H`, a handle to the current object.

#### Example



## 22 Graphics functions

### Graphics functions

- [ftitle](#) P377 - Add title to figure window.
- [highlight](#) P378 - Highlight specified range or date range in a graph.
- [maxfigure](#) P379 - Create graphics window stretched across the screen.
- [movetosubplot](#) P379 - Move an existing axes object or legend to specified subplot position.
- [plotcircle](#) P380 - Draw a circle or disc.
- [plotpp](#) P383 - Plot prior and/or posterior distributions and/or posterior mode.
- [plotmat](#) P381 - Visualise 2D matrix.
- [plotneigh](#) P382 - Plot local behaviour of objective function after estimation.
- [vline](#) P385 - Add vertical line with text caption at the specified position.
- [zeroline](#) P386 - Add zero line if Y-axis limits include zero.

### Getting on-line help on qreport functions

```
help grfun  
help grfun/function_name
```

---

### ■ ftitle

Add title to figure window

#### Syntax

```
AA = grfun.ftitle(TITLES,...)  
AA = grfun.ftitle(FF,TITLES,...)
```

### Input arguments

- FF [ numeric ] - Handle to graphical window.
- TITLES [ cellstr | char ] - Text string to be centred, or cell array of strings to be placed on the LHS, centred, and on the RHS of the figure.

### Output arguments

- AA [ numeric ] - Handle or handles to annotation objects.

### Description

### Example

---

## ■ highlight

Highlight specified range or date range in a graph

### Syntax

```
[PT,CP] = highlight(RANGE,...)
[PT,CP] = highlight(AX,RANGE,...)
```

### Input arguments

- RANGE [ numeric ] - X-axis range or date range that will be highlighted.
- AX [ numeric ] - Handle(s) to axes object(s) in which the highlight will be made.

### Output arguments

- PT [ numeric ] - Handle to the highlighted area (patch object).
- CP [ numeric ] - Handle to the caption (text object).

### Options

- 'caption=' [ char ] - Annotate the highlighted area with this text string.
- 'color=' [ numeric | [0.9,0.9,0.9] ] - An RGB color code or a Matlab color name.
- 'excludeFromLegend=' [ true | false ] - Exclude the highlighted area from legend.
- 'hPosition=' [ 'center' | 'left' | 'right' ] - Horizontal position of the caption.
- 'vPosition=' [ 'bottom' | 'middle' | 'top' | numeric ] - Vertical position of the caption.

### Description

### Example

---

## ■ maxfigure

Create graphics window stretched across the screen

### Syntax

```
fig = maxfigure(...)
```

### Output arguments

- fig [ numeric ] - Handle to the figure created.

### Options

See help on standar figure for the options available.

---

## ■ movetosubplot

Move an existing axes object or legend to specified subplot position

### Syntax

```
AX = movetosubplot(AX,M,N,P)
```

### Input arguments

- AX [ numeric ] - Handle to an existing axes object or legend.
- M, N, P [ numeric ] - Specification of the new position; see help on standard subplot.

### Output arguments

- AX [ numeric ] - Handle to the axes or legend moved to the new position.
- 

## ■ plotcircle

Draw a circle or disc

### Syntax

```
H = grfun.plotcircle(X,Y,RAD,...)
```

### Input arguments

- X [ numeric ] - X-axis location of the centre of the circle.
- Y [ numeric ] - Y-axis location of the centre of the circle.
- RAD [ numeric ] - Radius of the circle.

### Output arguments

- H [ numeric ] - Handle to the line or the filled area.

### Options

- 'fill=' [ true | false ] - Switch between a circle ('fill=' false) and a disc ('fill=' true).

Any property name-value pair valid for line graphs.

### Description

### Example

---

## ■ plotmat

Visualise 2D matrix

### Syntax

```
[POSH,NEGH,NANINFH,MAXH] = grfun.plotmat(X,...)
```

### Short-cut syntax

```
[POSH,NEGH,NANINFH,MAXH] = plotmat(X,...)
```

### Input arguments

- X [ numeric ] - 2D matrix that will be visualised; ND matrices will be unfolded in 2nd dimension before plotting.

### Output arguments

- POSH [ numeric ] - Handles to discs displaying non-negative entries.
- NEGH [ numeric ] - Handles to discs displaying negative entries.
- NANINFH [ numeric ] - Handles to NaN or Inf marks.
- MAXH [ numeric ] - Handles to circles displaying maximum value.

**Options**

- 'colNames=' [ char | cellstr | empty ] - Names that will be given to the columns of the matrix.
- 'rowNames=' [ char | cellstr | empty ] - Names that will be give to the row of the matrix.
- 'maxCircle=' [ true | false ] - Display a circle denoting the maximum value around each entry.
- 'scale=' [ numeric | 'auto' ] - Maximum value (positive) relative to which all matrix entries will be scaled; by default the scale is the maximum entry in the input matrix, `max(max(abs(X(isfinite(X)))))`.

**Description****Example**

---

**■ plotneigh**

Plot local behaviour of objective function after estimation

**Syntax**

```
[FIGH,AXH,OBJH,LIKH,ESTH,BH] = grfun.plotneigh(D,...)
```

**Input arguments**

- D [ struct ] - Structure describing the local behaviour of the objective function returned by the [neighbourhood](#) P109 function.

**Output arguments**

- FIGH [ numeric ] - Handles to the figures created.
- AXH [ numeric ] - Handles to the axes objects created.
- OBJH [ numeric ] - Handles to the objective function curves plotted.
- LIKH [ numeric ] - Handles to the data likelihood curves plotted.

- ESTH [ numeric ] - Handles to the actual estimate marks plotted.
- BH [ numeric ] - Handles to the bounds plotted.

### Options

- 'plotObj=' [ true | false ] - Plot the local behaviour of the overall objective function.
- 'plotLik=' [ true | false ] - Plot the local behaviour of the data likelihood component.
- 'plotEst=' [ true | false ] - Mark the actual parameter estimate.
- 'plotBounds=' [ true | false ] - Draw the lower and/or upper bounds if they fall within the graph range.
- 'subplot=' [ 'auto' | numeric ] - Subplot division of the figure when plotting the results.
- 'linkAxes=' [ true | false ] - Make the vertical axes identical for all graphs.

### Description

The data log-likelihood curves are shifted up or down by an arbitrary constant to make them fit in the graph; their curvature is preserved.

### Example

---

## ■ plotpp

Plot prior and/or posterior distributions and/or posterior mode

### Syntax

```
[PRG,POG,FIG,AX,PRLIN,POLIN,BLIN,PRANN] = grfun.plotpp(E,THETA,...)
[PRG,POG,FIG,AX,PRLIN,POLIN,BLIN,PRANN] = grfun.plotpp(E,ST,...)
[PRG,POG,FIG,AX,PRLIN,POLIN,BLIN,PRANN] = grfun.plotpp(E,PEST,...)
```

### Input arguments

- E [ struct ] - Estimation input struct, see [estimate](#) P76, with prior function handles from the [logdist](#) P155 package.
- THETA [ numeric | empty ] - Array with the chain of draws from the posterior simulator [arwm](#) P148.
- ST [ struct | empty ] - Output struct returned by the posterior simulator statistics function [stats](#) P152.
- PEST [ struct | empty ] - Output struct returned by the [model/estimate](#) P76 function containing the posterior mode estimates.

### Output arguments

- PRG [ struct ] - Struct with x- and y-axis coordinates to plot the prior distribution for each parameter.
- POG [ struct ] - Struct with x- and y-axis coordinates to plot the posterior distribution for each parameter.
- FIG [ numeric ] - Handles to the figures created.
- AX [ numeric ] - Handles to the axes (graphs) created.
- PRLIN [ numeric ] - Handles to the prior lines plotted.
- PRLIN [ numeric ] - Handles to the posterior lines plotted.
- BLIN [ numeric ] - Handles to the lower and upper bound lines plotted.
- TIT [ numeric ] - Handles to the graph titles created.

### Options

- 'describePrior=' [ 'auto' | true | false ] - Add one extra line to each graph title describing the prior (name, mean, and std dev); if 'auto=' the description will be shown only if 'plotPrior=' is true.
- 'plotPrior=' [ true | false ] - Plot prior distributions.
- 'plotPoster=' [ true | false ] - Plot posterior distributions.
- 'plotBounds=' [ true | false ] - Add lower and/or upper bounds to the distribution graphs; if false, the bounds are only added if they are within the graph x-limits.



- `'sigma='` [ numeric | 3 ] - Number of std devs from the mean or the mode (whichever covers a larger area) to the left and to right that will be plotted unless running out of bounds.
- `'tight='` [ true | false ] - Make graph axes tight.
- `'xLims='` [ struct | empty ] - Control the x-limits of the prior and posterior graphs.

## Description

If you call `plotpp` with `PEST` (i.e. a struct with posterior mode estimates) as the second argument, the posterior modes are plotted as vertical lines (stem graphs).

## Example

---

## ■ `vline`

Add vertical line with text caption at the specified position

## Syntax

```
[LN,CP] = grfun.vline(AX,POSITION,...)
[LN,CP] = grfun.vline(POSITION,...)
```

## Input arguments

- `AX` [ numeric ] - Handle to an axes object in which the vline will be placed.
- `POSITION` [ numeric ] - Horizontal position on the x-axis at which the vline will be placed; in tseries graphs this is supposed to be a date.

## Output arguments

- `LN` [ numeric ] - Handle to the vline(s) plotted (line objects).
- `CP` [ numeric ] - Handle to the caption(s) created (text objects).

## Options

- 'caption=' [ char ] - Annotate the vline with this text string.
- 'excludeFromLegend=' [ true | false ] - Exclude the vline from legend.
- 'hPosition=' [ 'center' | 'left' | 'right' ] - Horizontal position of the caption.
- 'vPosition=' [ 'bottom' | 'middle' | 'top' | numeric ] - Vertical position of the caption.
- 'timePosition=' [ 'after' | 'before' | 'middle' ] - Placement of the vertical line on the time axis: in the middle of the specified period, immediately before it (between the specified period and the previous one), or immediately after it (between the specified period and the next one).

## Description

### Example

---

## ■ zeroline

Add zero line if Y-axis limits include zero

## Syntax

```
LN = zeroline(...)  
LN = zeroline(AX,...)
```

## Input arguments

- AX [ numeric ] - Handle to an axes object (graph) to which the zeroline will be added; if not specified the current axes will be used.

## Output arguments

- LN [ numeric ] - Handle to the line plotted (line object).

### **Options**

Any options valid for the standard plot function.

### **Description**

### **Example**